# Mündəricat


BÖLMƏ 1. C# proqramlaşdırma dili ..... 3
C# proqramlaşdırma dilinə giriş ..... 3
Visual Studio IDE ..... 5
Ilk Bəsit Proqram ..... 8
İkinci Bəsit Proqram ..... 12
If şərt ifadəsi ..... 14
for dövr ifadəsi ..... 17
BÖLMƏ 2. Verilənlər tipləri və Dəyişənlər ..... 19
Verilənlər tipi, tip nədir? ..... 19
Dəyişən anlayışı ..... 20
Mənimsətmələrə yaxından baxış ..... 24
Dəyişənlərin yaşama müddəti ..... 25
Tip Çevrilmələri və Tip Mənimsətmələr ..... 27
Avtomatik Tip çevrilmələri ..... 27
Açıq tip mənimsətmələr ..... 27
Manual Tip Çevrilmələri ..... 29
İdentifikatorlar ..... 33
Bəzi İpucları və Qaydalar ..... 33
BÖLMƏ 3. OPERATORLAR ..... 35
Operator nədir? ..... 35
Vahid operatorlar ..... 35
Cəbri operatorlar ..... 38
Müqayisə Operatorları ..... 41
Məntiqi Operatorlar ..... 42
Şərtə Bağlı Məntiqi Operatorlar ..... 46
Mənimsətmə operatoları ..... 48
Bitişik mənimsətmələr ..... 48
Bit Əsaslı Operatorlar ..... 49
? Operatoru ..... 54

Operatorların Öncəlik Sırası ..... 55
BÖLMƏ 4. PROQRAM KONTROL İFADƏLƏRİ ..... 57
if Şərt İfadəsi ..... 57
İç-içə if ifadələri ..... 58
if-else-if kombinasiyası ..... 60
switch İfadəsi ..... 62
for dövr ifadəsi ..... 65
Birdən çox dövr idarəedən dəyişən ..... 66
Sonsuz dövr ..... 68
while dövr ifadəsi ..... 68
do-while dövr ifadəsi ..... 70
break ifadəsi ..... 71
continue ifadəsi ..... 73
BÖLMƏ 5. OBYEKT YÖNÜMLÜ POQRAMLAŞDIRMA, SİNİFLƏRƏ, METODLARA OBYEKTLƏRƏ GİRİŞ ..... 75
Obyekt yünümlü anlayışı. ..... 76
Abstraksiya (Abstraction) ..... 77
İnkapsulyasiya (Encapsulation) ..... 77
Hüquq təyinedicilər (Access Modifiers) ..... 78
Siniflərin əsasları ..... 78

# BÖLMƏ 1. C# proqramlaşdırma dili

## C# proqramlaşdırma dilinə giriş

Əvvəlcə qeyd edək ki, bu proqramlaşdırma dilinin adı C# - dır (C Sharp), və "si-şarp" kimi tələffüz olunur.

Əvvəlcə .NET Framework – un nə olduğunu qısa şəkildə aydınlaşdıraq. 2000-ci ilə qədər, proqramçı olmaq istəyən şəxslər, hansı proqramlaşdırma dilini seçməli olduqları haqqda ciddi tərəddüdlə qarşılaşırdılar. Çünki seçilən texnologiyanın mühitdən asılılığı, edəcəkləri işi, seçdikləri proqramlaşdırma dilinin nə dərəcədə ələ ala biləcəyi kimi faktorları nəzərə almaq lazım idi. Üstəlik, çoxlu sayıda fərqli sistemdən və fərqli arxitekturaya malik kompüterlərdən ibarət internet dediyimiz virtual dünyada, veb

tətbiqetmələr yazmaq lazım olduğunda, dillərin mühitdən asılılığı, proqramçıları lap boğaza yığırdı. Mühitdən asılılıq dedikdə, bir proqramlaşdırma dilində yazılan proqramın, sadəcə spesifik sistemlərdə və spesifik prosessorlar tərəfindən icra olunmağı başa düşülürdü. Fikirləşin, bir proqram yazırsınız, bu proqram ancaq bir sistemdə sadəcə müəyyən sayda prosessorlar tərəfindən icra oluna bilər. Başqa sistemə proqramımızı yazmaq lazım gəldikdə, gərək proramımızı sistemə və prosessora görə yenidən kompilyasiya edək. Bu məsələ, 2000-ci ilə qədər ciddi bir problem idi. Bunun üzərinə 1991-ci ildə Sun Microsystems şirkəti çox güclü və mühitdən bağımsız bir dil olan Java – nın əsasını qoydu. "Mühitdən bağımsız" sözünü eşidən bəzi proqramçılar (söhbət 90-cı illərin sonlarından gedir) qulaqlarına inanmadılar. Bu azadlıq idi! Java ilə yazılan proqramlar, istənilən sistemdə və prosessorlarda işləyə biləcəkdi. Bunun üzərinə kütləvi şəkildə Java – ya axış yaranmağa başladı və qısa müddət arzində çox sevilən bir dil oldu. Deməli, Java dilinin qurucuları incə bir məntiqlə mühitdən asılılıq problemini demək olar ki aradan qaldırdı. Belə ki, Java proqramlarının kodları birbaşa maşın dilinə yox, "bytecode" adlanan xüsusi bir aralıq dilə çevrilirdi. Bytecode – dan ibarət proqram isə Java Virtual mühərriki (JVM – Java Virtual Machine) olan istənilən sistemdə işləyə biləcəkdi. JVM, həmin bu bytecode – u yerləşdiyi sistemə və prosessora uyğun şəkildə yenidən kompiylasiya edərək maşın dilinə çevirir və beləliklə mühitdən asılılıq aradan qaldırılır. Java kimi gözəl bir dilin yaranması, Bill Gates – i dəli edir. Əsəbləşən Bill Gates tez Anders Hejlsberg – i yanına çağırır və "nə edirsiz edin, tez mənə Java kimi gözəl bir dil yaradın", - deyir. Əslində, bu ifadə ilə C# -a haqqsızlıq etmiş oluruq. C# - ın yaranması tam olaraq Java – nın meydana gəlməsində yox, 100% .NET Framework dəstəkli bir dilin olması zərurətindən irəli gəldi. İndi məsələni başdan alaq. Deməli, 90 – larda proqramlaşdırma dili seçimi qarşısında qalmaq, böyük qərarsızlığa səbə olurdu. Çünki, tək bir proqramlaşdırma dili, edilən işi tam şəkildə mükəmməl ələ ala bilmirdi. Yəni, yüksək bir proyektin əsası qoyulduğunda vəziyyət elə yerə gəlirdi ki, "proyektin filan hissəsini filan dildə, başqa hissəsini də ona uyğun dildə yaza bilsəydik nə gözəl olardı" fikirləri dərd olmuşdu. Bunun üzərinə, Microsoft şirkəti .NET Framework adlı bir işləmə mühiti yaratdı və mövcud bir çox proqramlaşdırma dilini bu mühitlə uyğunlaşdırdı. Bu o deməkdir ki, artıq eyni bir işi bir neçə proqramlaşdırma dilində görə biləcək. Çünki, .NET Framework uyumlu dil dedikdə, kitabxanalarını .NET Framework – dan götürən və və müəyyən standartlara cavab verən (bu standartlara CLI – Common Language Infrastructure deyilir) bir dil başa düşülür. Beləliklə, .NET Framework mühitində, bir proqramçı bir layihəni hissələrə bölərək müvafiq hissə üçün ən uyğun dili seçib onunla işləyə bilər, sonra hissələri birləşdirib yekun nəticəni təqdim edə bilər. Bu işi yerinə yetirmək üçün, məhz .NET Framework mühiti yaradıldı. Dünyadakı bir çox populyar proqramlaşdırma dilləri də, .NET Framework üçün optimizasiya edildi. Məsələn, C, C++, VisualBasic,, Jscript, ADA, Perl, Python, Smalltalk, Pascal, Haskel, Eiffel, COBOL və s. Amma, nə qədər olmasa da bir proqramlaşdırma dilinin .NET Framework üçün

100% uyumlu versiyasını çıxarmaq, mümkün deyildi. Çünki, dilin strukturunu kökündən dəyişmək olmazdı. Buna görə də Microsoft şirkəti .NET Framework mühitini 100% dəstəkləyən iki proqramlaşdırma dili çıxardı və bu dillərə C# və VB.NET adını verdi. Yəni, C# dili, .NET Framework mühitində daha original və proqressiv işləmək üçün yaranmış bir dildir əslində. Bu kimi .NET Framework uyumlu dillərə qarışıq proqramlaşdırma dili (mixed programming language) dəstəyi olan dil deyilir.

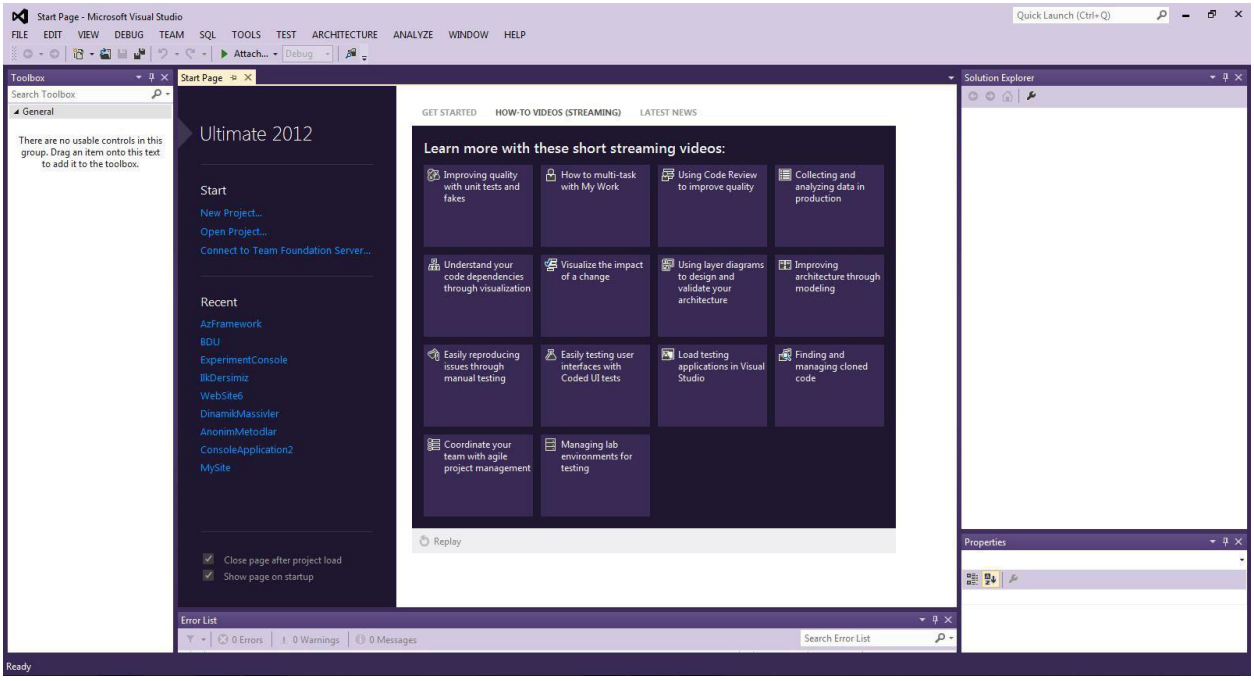
C# dilini yaradanların başında dünyanın ən güclü proqramlaşdırma dili mütəxəssislərindən biri olan Anders Hejlsberg dayanır. Bu adam, 60-cı illərdə çox populyar dil olan Turbo Pascal – ın original qurucusudur.

## Visual Studio IDE

Proqram yazmaq üçün, bir proqramçıya əslində 3 şey lazımdır:

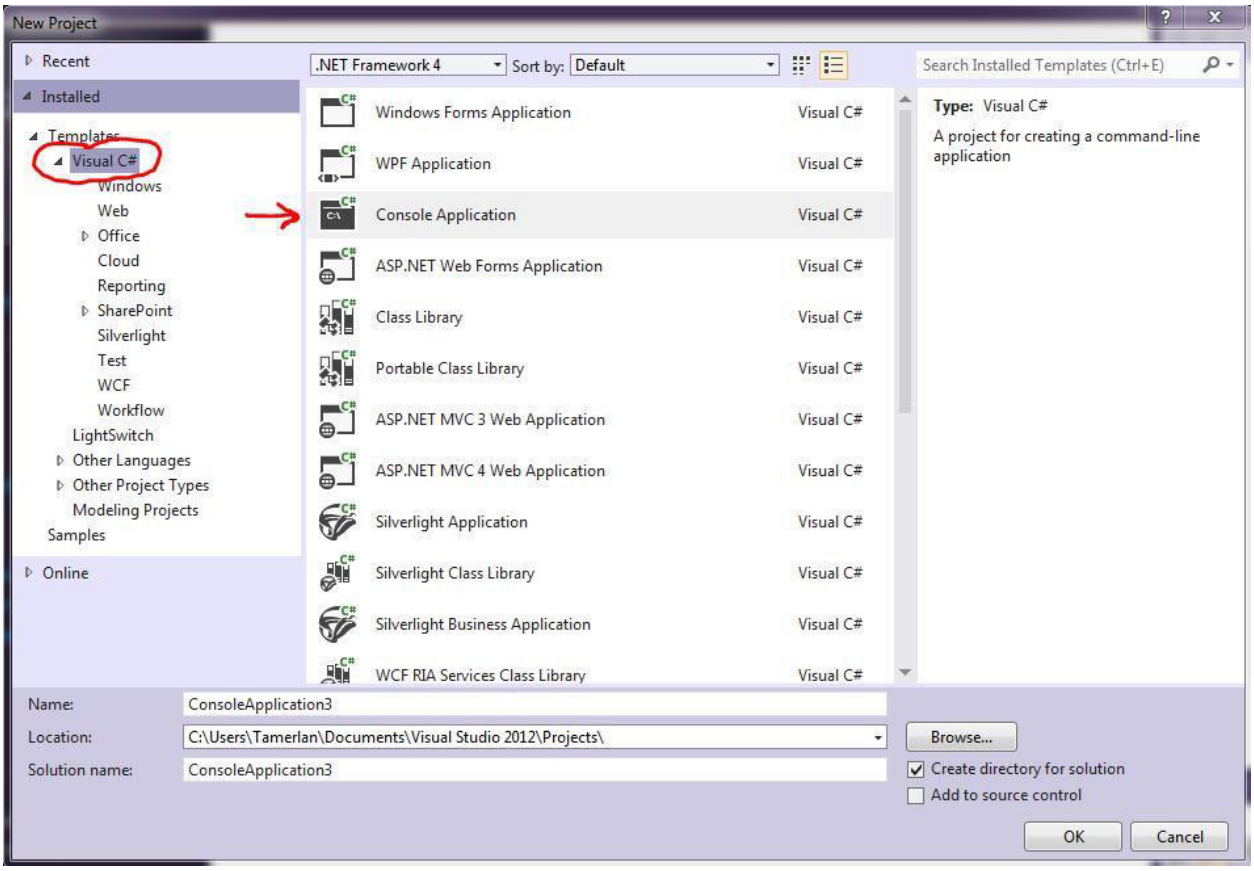
1. Bir mətn redaktoru
2. Müvafiq proqramlaşdırma dilinin kompilyatoru
3. Kompilyasiyadan alınan nəticəni icra edən, sistemə yüklü işləmə mühərriki (CLR).
   Amma real dünyada proqram yazmaq üçün müəyyən vasitələrdən istifadə edirlər. Adi bir mətn redaktorunda, məsələn hamımıza doğma Notepad- da bir bəsit proqram yazdığımızı düşünək. Bu zaman tutaq ki, 4 ədəd sintaktik səhv etmişik və bundan xəbərimiz yoxdur. İndi həmin proqram kodunu, icra edə biləcəyimiz *.exe genişlənməsinə malik bir fayla çevirməliyik, yəni kodu kompilyasiya etməliyik.

Kompilyasiya edəndə kompilyator həmin 4 dənə sintaktik səhvə görə kodların kompilyasiyasını başa çatdıra bilməyəcək və terminal pəncərəsində filan sətrin filan sütununda xəta var deyəcək. Bu zaman filan sətrin filan sütununu axtaracağıq və xətanı aradan qaldırıb kodları yenidən kompilyasiya etməyə çalışacağıq. Lap tutaq ki, proqramımız işlədi, çox gözəl. Bəs uzun bir proyekt yazmaq istəsəm, çox demirəm 100 sətirlik bir C# proqramını NotePad ilə yazmağa cəhd etmək tam olaraq dəlilikdir. Buna görə də proqram yazmaq üçün yüksək səviyyəli mətn redaktorlarından istifadə olunur hansı ki, bu redaktorlar kodlardakı sintaktik səhvləri avtomatik aşkar edir, xətaların altından qırmızı xətt çəkir və bu vəziyyətdə kodları kompilyasiya etməyə icazə vermir. Bu redaktorların işi ancaq bununla bitmir, bizə vizual olaraq proqramşaldırma etməyə də imkan verir, proqram fayllarını tam nəzarətdə saxlayır, kodlarımıza hakim çıxır və işimizi dəfələrlə asanlaşdırır və daha nələr nələr... Bax bu kimi, proqram yazmaq üçün istifadə olunan mətn redaktorlarına deyilir IDE – Integrated Development Environment. Məsələn, Microsoft Visual Studio, Adobe Dreamweaver, Java Eclipse və s. Biz də C# dilində proqramları Visual Studio IDE - si ilə yazacağıq. Proqramı http://www.microsoft.com adresindən yükləyə bilərsiniz. Proqramı kompüterə quraşdırmaq adi bir proqramı quraşdırmaq kimi asandır, sadəcə lisenziya razılaşmasını qəbul edirsiniz və lazımi komponentlərin sistemə qurulmasını gözləyirsiniz. Proqramımızı kompüterə quraşdırdıqdan sonra onu icra edin, aşağıdakı kimi bir pəncərə ilə qarşılaşacaqsınız:



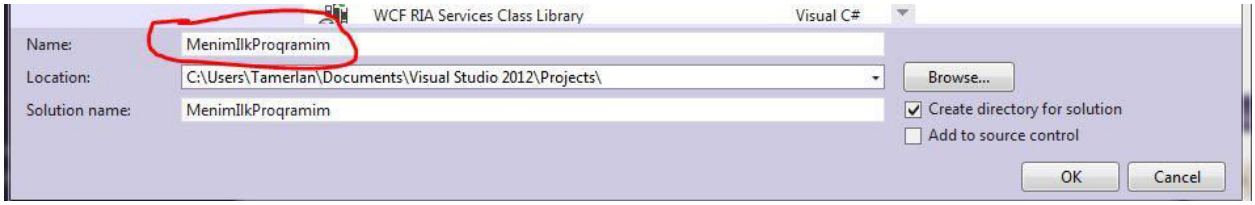

Bu pəncərəyə Başlanğıc Səhifə (Start Page) deyilir. Burada Visual Studio – nun yenilikləri ilə tanış ola bilərsiniz və s. Proqram yazmaq üçün əvvəlcə müvafiq işə

uyğun proyekt yaratmalıyıq. Bir proyekt yaraqmaq üçün ya həmin bu başlanğıc səhifəsindən "New project…" klikləyirsiniz, ya da File >> New >> Project yolunu izləyirsiniz. Açılan pəncərədə müxtəlif proyekt nümunələrini görə bilərsiniz. Qeyd edək ki, Visal Studio – nun hansı nəşrini istifadə etməyinizdən asılı olaraq, bu siyahıda göstərilənlərin sayı sizdə fərqli ola bilər.




Kənardan "Visual C#" – ın optimizasiya olunduğuna diqqət edin. Əgər VS – ni ilk dəfə işə salırsınızsa, açılan pəncərədə sizə hansı proqramlaşdırma dilini seçməyiniz barədə seçim imkanı gəlir. Orada Visual C# - ı seçdiyinizdən əmin olun. Yuxarıdakı siyahı onu ifadə edir ki, C# proqramlaşdırma dili bizə bu proyektləri həyata keçirməyə imkan verir. Məsələn Windows sistemləri üçün ən məşhur proqramlaşdırma texnologiyası Windows Form Application, daha yüksək vizual qrafikaya malik proqramlar (məsələn 3D oyunlar) yazmaq üçün WPF – Windows Presentation Foundation Application, Console Application, Veb proqramlaşdırma üçün ASP.NET Web Forms Application ya da Microsoft – un yeni nəsil Veb proqramlaşdırma texnologiyası olan ASP.NET MVC Web Application kimi proyektləri hazırlaya bilərik. İstənilən proqramlaşdırma dili ən sadə struktura malik olan proqram növündə - konsolda öyrənilməyə başlayır. Eyniylə biz də C# dilini Console Application üzərindən öyrənəcəyik. Bir konsol proyekt yaratmaq üçün Console Application üzərində klikləyirsiniz və aşağıdan proyektinizə ad verirsiniz. Məsələn,

"MenimIlkProqramim" adlı bir konsol proyekt yaradaq və OK klikləyək:




Beləliklə biz, Visual Studio ilə ilk proyektimizi yaratmış olduq. Hamızını təbrik edirəm!

## Ilk Bəsit Proqram

Artıq bir proqram yazmağın vaxtı gəldi, çatdı. Yuxarıda göstərilən qaydada bir konsol proyekt yaratdığınızda Visual Studio sizə mətn redaktə pəncərəsini aşağıdakı kodlarla birlikdə təqdim edəcək:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MenimIlkProqramim
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Bu kodları aşağıdakı kimi dəyişdirin:

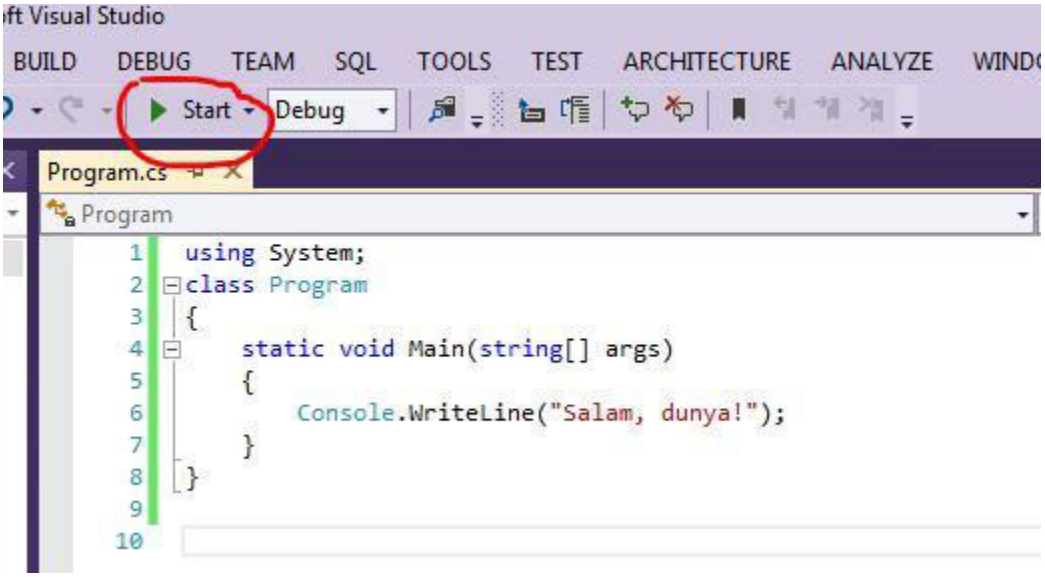
```
using System;
class Program
{
    static void Main()
    {
    }
}
```

Kodlardakı bəzi hissələri sildik, çünki bizə hələki lazım olmayan bölümləri təmizlədik. Çox kod gözünüzü qorxutmasın deyə. Bu bir neçə sətir kod, ən bəsit şəkildə proqram yazağa hələki kifayət quruluşa malikdir. İndi izahata keçməmiş, Main() metodunun (metod nə olduğunu hələki bilmirik) gövdəsinə Console.WriteLine("Salam, dunya!"); sətrini əlavə edək. Kodlar aşağıdakı kimi olacaq:

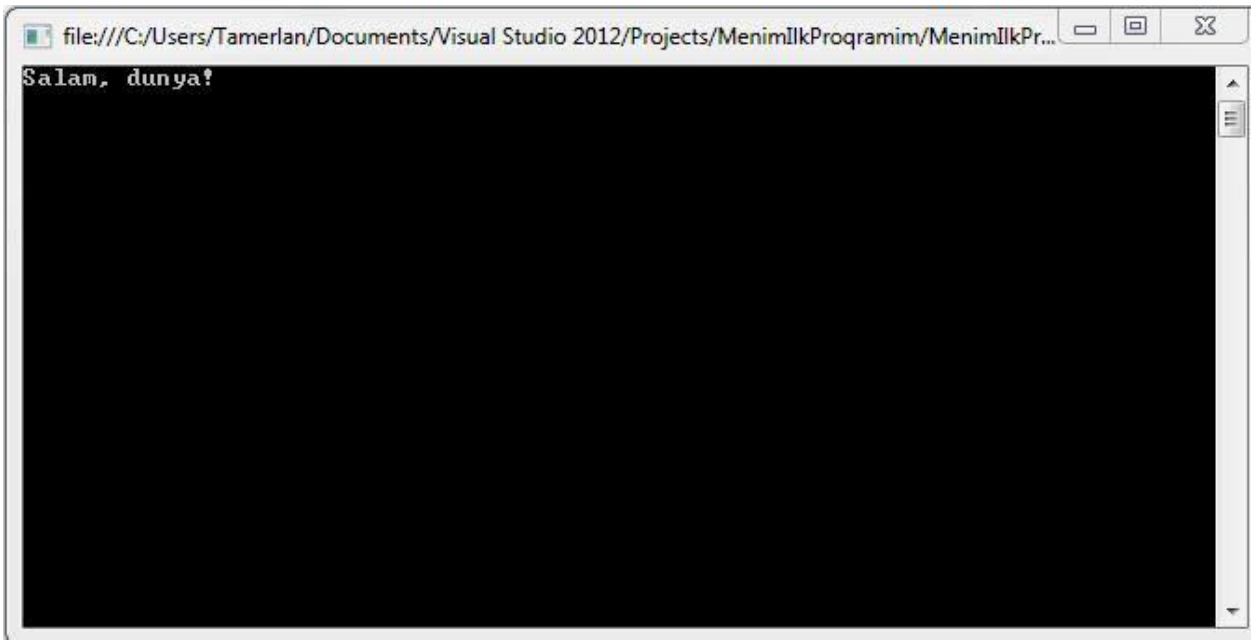
```
using System;
class Program
{
    static void Main()
```

```
    {
        Console.WriteLine("Salam, dunya!");
        Console.ReadKey();
    }
}
```

Bu bizim ilk proqram kodları, qaldı bunu icra etmək. Bunun üçün qeyd etdiyimiz kimi, kodları sistemin tanıyacağı bir şəklə gətirmək, yəni kompilyasiya etmək. Bunun üçün kodlaşdırma pəncərəsinin yuxarı hissəsindəki "Start" düyməsini kliklləməyiniz kifayətdir.




Eyni işi yuxarıda Debug >> Start Debugging yolunu izləyərək də edə bilərsiniz. Beləliklə proqramımız kompilyasiya olunur və nəticədə *.exe faylı yaranız. Visual Studio yaranmış bu faylı avtomatik işə salır. Nəticədə aşağıdakı kimi bir pəncərə özünü göstərir:




Təbriklər! Bu bizim ilk proqramımızdır. İndi başlayaq yuxarıdakı bəsit proqramı izah etməyə. Əvvəlcə qeyd edim ki, C# dili tamamilə obyekt yönümlü bir dildir. Buna görə də, C# -da bir şeyi bilməmiş, digərini tam olaraq başa düşmək mümkün deyil. Ona görə də, əgər proqramlaşdırmada yeni isəniz, bu proqramı 100% başa düşməyəcəksiniz. Bunun üçün narahat olmayın. Deməli, əvvəlcə gəlir

```
using System;
```

deyə bir sətir. Bu ifadə System adlı kitabxananı proqramımıza qoşur. Kitabxana dedikdə, öz içində sinifləri və digər ad fəzalarını (namespaces) saxlayan fiziki olaraq *.dll genişlənməsinə malik fayl başa düşülür. Bu fayl *C:|Windows|Microsoft.NET|Framework64|v4.0.30319* ünvanında yerləşir (System.dll). Bu kitabxana, .NET Framework – un baza sinif kitabxanasıdır. Bu kitabxananın içində .NET – in verilənlər tipləri, standart sinifləri o cümlədən Console sinfi yerləşir. Ona görə də kitabxananı proqrama qoşmalıyıq. Sonra

```
class Program
{
```

deyə bir sətir gəlir. Bu sətir Program adlı bir sinif təyin edir. Qeyd olunduğu kimi C# tamamilə obyekt yönümlü bir dildir olduğu üçün, hər şey bir sinfin içərisində olmalıdır. Ona görə də bəsit də olsa bir sinif yaratdıq. Sonra isə açılan fiqurlu mötərizə "{" sinfin gövdəsinin açılması anlamına gəlir. Buna uyğun "}" mötərizəsi arasındakı hissələr sinfin bir üzvüdür. Məsələn növbəti gələn

```
static void Main()
{
```

sətri Main adlı bir metod təyin etməyə uyğundur. Bu metod proqramın başlanğıc nöqtəsidir və hər C# proqramında hökmən bir dənə və yalnız və yalnız bir dənə **static** keyfiyyətə malik Main adlı metod olmalıdır. Buradakı **static** sözü, Main metodunun əməliyyat sistemi tərəfindən çağrıldığına görə qeyd olunmalıdır. Static sözünü silib proqramı icra etməyə çalışsanız, xəta ilə qarşılaşacaqsınız. **void** isə metodun heç bir

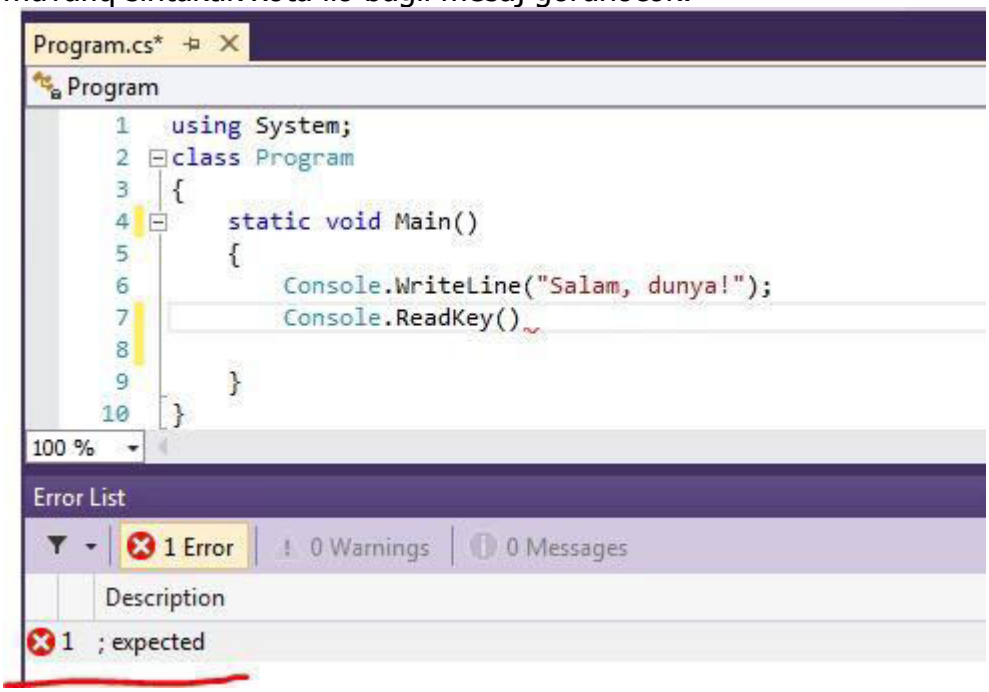
qiymət qaytarmadığı anlamına gəlir. Yəni bu metod, müəyyən əməliyyat yerinə yetirir, amma nəticədə bizə qiymət olaraq bir şey vermir.

**DİQQƏT!** C#, böyük-kiçik hərfə həssas bir dildir, yəni C# -da "a" ilə "A" ayrı-ayrı şeylərdir. Buna görə də Main metodunun adını balaca hərflə yazıb (main) proqramı icra etməyə çalışdığınızda xəta mesajı alacaqsınız. Bu terminologiyanın ingiliscə adı "Case Sensitive" – dir.

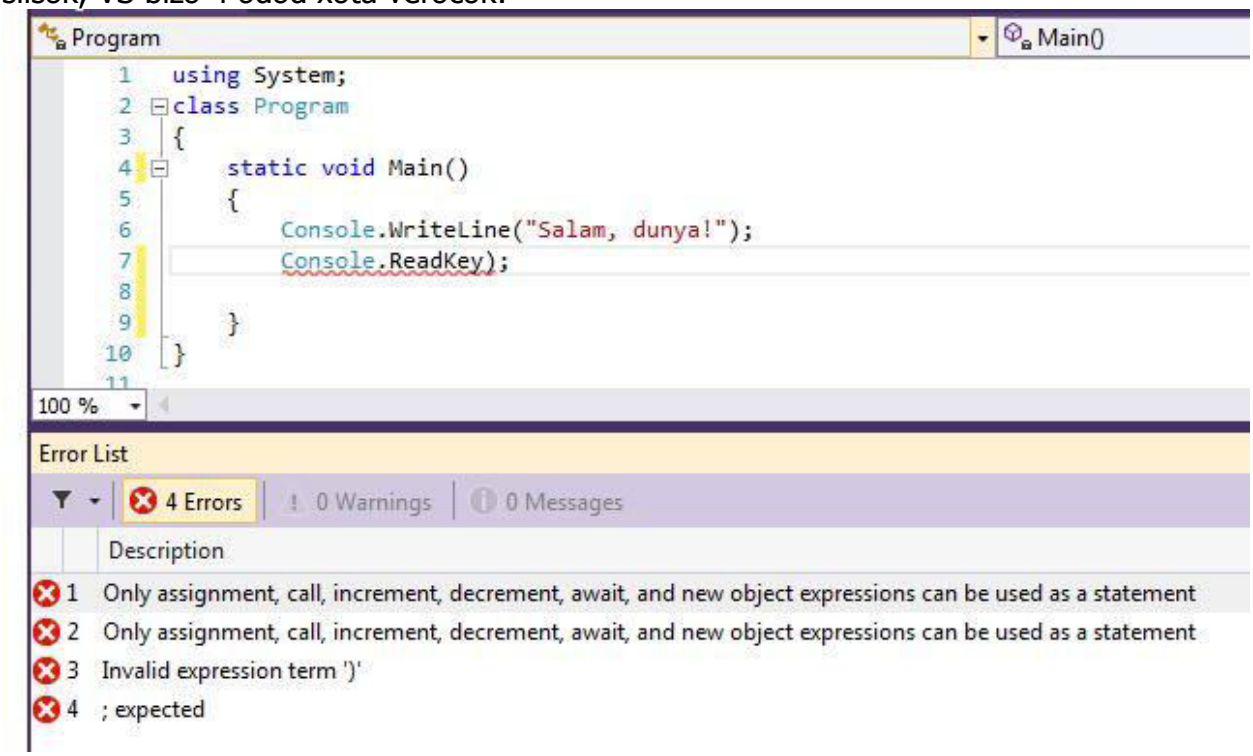
Sonra gəlir
**Console.WriteLine("Salam, dunya!");**
sətri. Bu sətir Console sinfindəki WriteLine() metodunun "Salam, dunya!" arqumenti ilə çağrılmasına uyğundur. Hələki belə başa düşək: Console.WriteLine() metodu, mötərizələrin içinə, dırnaq işarələri ilə yazılan sözləri ekrana çıxarmaq üçün istifadə olunur. (Əslində, arxa planda gedən əməliyyat kifayət qədər mürəkkəbdir). Beləliklə, ekrana "Salam, dunya!" ifadəsi çıxır.

Konsol proqramlarda əməliyyatlar icra olunduqdan sonra, proqram avtomatik bağlanır. Ona görə də konsol pəncərələrini, ekranda saxlamaq üçün müəyyən üsullardan istifadə edirlər. Növbəti gələn
**Console.ReadKey();**
ifadəsi, bunu təmin etmək üçündür. Yəni, proqram ekrana gələn kimi itməsin, hər hansısa bir düymə basılana qədər gözləsin.

Beləliklə, bu bizim ilk proqramımız oldu. Deməli, Microsoft Visual Studio – nin bir IDE olduğunu qeyd etmişdik. Mahiyyət etibarı ilə, Visual Studio (qısaca VS) əslində çox güclü bir mətn redaktorudur, biz proram yazanda kodlarımıza nəzarət edir, sintaktik xıtaları aşkarlayır və bunu bizə bildirir. Məsələn, **Console.ReadKey() –** dən sonrakı ";" işarəsini silsək, VS həmin nöqtənin altında qırmızı xətt çəkəcək və Error List bölməsində müvafiq sintaktik xəta ilə bağlı mesaj görünəcək:



"; expected", yəni "; gözlənilmədi". Əgər, Error list sizdə görünmürsə, yuxarıda VIEW panelindən Error List naviqasiyasını seçin və ya Ctrl + W + E kombinasiyasını sıxın. Başa düşmək lazımdır ki, Visual Studio bir insan deyil və o, sintaktik xətaları kompilyatorun verdiyi mesajlara əsasən müəyyənləşdirir. Yəni, bəzən VS – in əks etdirdiyi xətalar düzgün olmaya bilər. Məsələn, **Console.ReadKey);** ifadəsindəki birinci mötərizəni silsək, VS bizə 4 ədəd xəta verəcək:




Göründüyü kimi, bircə dənə sintaktik səhv etməyimizə baxmayaraq, 4 ədəd xəta mesajı aldıq. Üstəlik, bu xətaların heç biri, müvafik siktaktik səhvi düzgün xarakterizə etmir. Yəni, VS xəta verdiyi zaman, xətanın baş verdiyi sətirlə yanaşı, ətrafındakı bir neçə sətirə də nəzər yetirin.

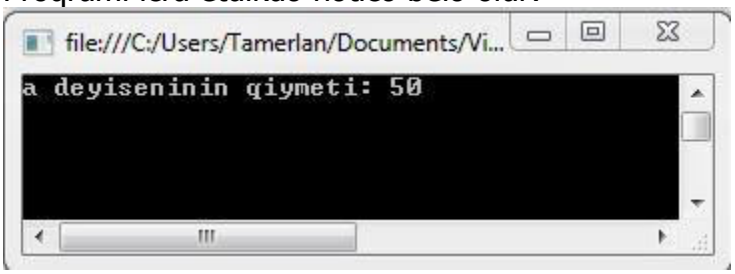
## İkinci Bəsit Proqram

İndi proqramda bir az dəyişiklik edək. Bir **dəyişən** təyin edək və buna qiymət **mənimsədək**. Bir dəyişənə qiymət mənimsətmə, proqramlaşdırma dillərindəki bəlkə də ən vacib əmliyyatlardan biridir. Dəyişənlər geniş bir şəkildə qeyd olunacaq. Hələki bəsit öyrənəcəyik. Dəyişən yaddaşın bir hissəsidir və sahib olduğu qiyməti bu yaddaş hissəsində saxlayır. Dəyişənin bir də **tipi** olur, tip dəyişənin nə kimi məlumatları özündə saxlayacağını təyin edir. Aşağıdakı proqrama baxaq:

```
using System;
class Program
{
    static void Main()
    {
        int a;                                                  (1)
        a = 50;                                                 (2)
        Console.WriteLine("a deyiseninin qiymeti: " + a);       (3)
```

```
        Console.ReadKey();
    }
}
```

Proqramı icra etdikdə nəticə belə olur:




(1) sətrindəki int a; ifadəsi bir dəyişən təyin edir və bu dəyişənə "a" adını verir. Bu dəyişənin tipi isə int- dir, yəni tam tip. Beləliklə yaddaşda (RAM) int tipinin xarakteristikalarına malik 4 bayt yer ayrılır və bu yaddaş sahəsinə "a" adı verilir. (2) sətrindəki a = 50; ifadəsi a dəyişəninə 50 qiymətini verir. Beləliklə, yaddaşın həmin hissəsinə 50 informasiyası yüklənir. (3) sətrində isə "a deyiseninin qiymeti" ifadəsini, ardından a dəyişənin qiymətini (yəni, 50 - ni) ekrana yazdırdıq. Diqqət etməyimiz gərəkən, nüans: Console.WriteLine("a deyiseninin qiymeti: " + a); kdunda, "a deyiseninin qiymeti: " ifadəsini ekrana yazdırdıqdan sonra + a ilə, proqramdakı a adlı identifikatora müraciət etmiş oluruq. Yəni, + operatorundan sağdakı a, bir sətir kimi ekrana çıxmır, proqramdakı a dəyişəninin qiymətini ekrana çıxarır. İndi iki dəyişən də təyin edək və bu dəyişənlərin qiymətlərindən birini, digərinə bölək:

```
using System;
class Program
{
    static void Main()
    {
        int a;
        int b;
        int c;
        a = 50;
        b = 10;
        c = a / b;
        Console.WriteLine("50 bolunsun 10 = " + c);
        Console.ReadKey();
    }
}
```

Nəticə aşağıdakı kimi olur:

Deməli, bu proqramda 3 dənə a, b və c adlı dəyişən təyin etdik. a dəyişəninə 50 qiymətini, b dəyişəninə 10 qiymətini mənimsətdik. C dəyişəninə isə a – nın b dəyişəninə bölünməsindən alınan qiyməti mənimsətdik, yəni c 5 dəyərinə sahib olur. Fikir versənir

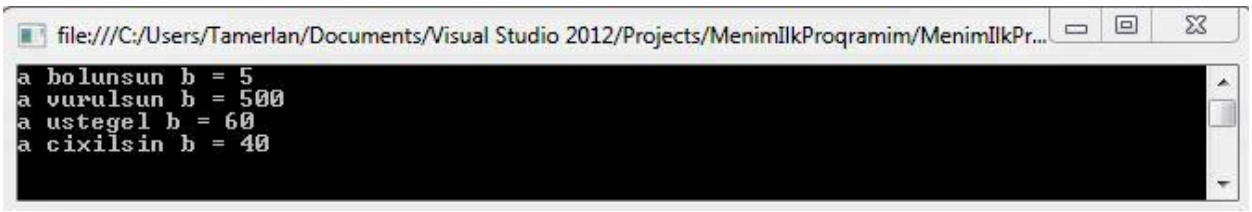
c = a / b; sətrində "/" deyə bir simvol var. Bu bir operatordur və vəzifəsi soldakı operandın qiymətini, sağdakına bölməkdir. Yəni a / b = 50 / 10 = 5 olur. Bu operator bir arifmetik operatordur, / ilə yanaşə digər arifmetik operatorlardan bəzilərini qeyd edək:

+ toplama
- çıxma
* vurma
/ bölmə

İndi bu operatorların istifadəsini göstərən bir proqrama baxaq:

```
using System;
class Program
{
    static void Main()
    {
        int a;
        int b;
        int c;
        a = 50;
        b = 10;
        c = a / b;                                      //c = 5
        Console.WriteLine("a bolunsun b = " + c);
        c = a * b;                                     //c = 500
        Console.WriteLine("a vurulsun b = " + c);
        c = a + b;                                     //c = 60
        Console.WriteLine("a ustegel b = " + c);
        c = a - b;                                     //c = 40
        Console.WriteLine("a cixilsin b = " + c);
        Console.ReadKey();
    }
}
```

Nəticə, aydındır ki, aşağıdakı kimi olacaq:




## If şərt ifadəsi

Biz proqramımızı işə salanda, proqramımız yuxarıdan aşağı doğru (Main metodundan başlayaraq) sətir-sətir icra olunmağa başlayır. Bəzən vəziyyət elə olur ki, proqramın müəyyən hissəsinin icra olunub – olmaması, hansısa şərtə bağlı olsun. Şəni müəyyən bir şərt daxlində poqramın bir hissəsi icra olunsun, ya da icra olunmasın. Bu kimi proqramın icra olunma axışını idarə edən ifadələr mövcuddur ki, bu ifadələrə proqram kontrol ifadələri (program control statements) deyilir. Bunlardan biri **if** – dir. If bir idarə etmə ifadəsidir və vəzifəsi ondan ibarətdir ki, hansısa bir şərtin doğru olduğu təqdirdə, hansısa kodlar icra olunsun, əks halda –

şərt düzgün olmadığı halda həmin kodlar icra olunmasın (if ifdəsini "proqram kontrol ifadələri" bölməsində geniş öyrənəcyik). Bu ifadənin sintaktik şəkli belədir:

```
if(şərt)
{
    //Əməliyyatlar
}
```

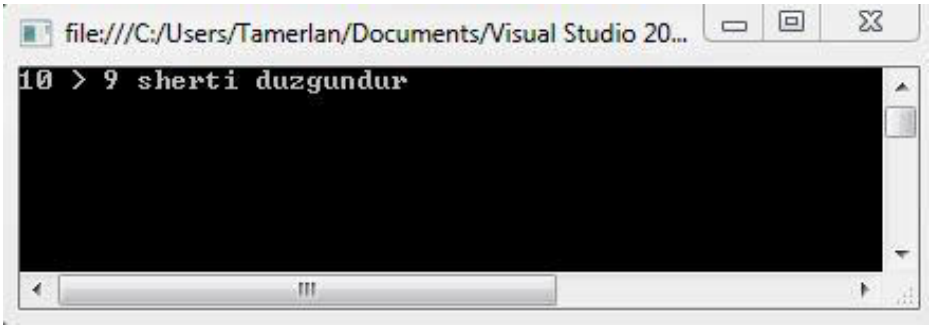
Burada şərt doğru olsa (true) müvafiq əməliyyatlar icra olunur. Əks halda əməliyyatlar icra olunmur. Burada **şərt** bool tipində bir qiymətdir. Bool tipinə aid dəyişənlər özündə iki qiymət saxlaya bilər: true, false. True doğru, false isə yalnış nəticəyə uyğundur. Yəni if ifadəsinin şərti true olarsa, bu ifadəyə aid kodlar icra olunacaq. Yalnış olsa, yəni false, gövdə icra olunmayacaq. Qeyd edək ki, əgər if ifadəsinin şərti düzgün olduğu halda yerinə yetiriləcək sadəcə bir kod sətri varsa, onda fiqurlu mötərizələr yazılmaya da bilər. Yəni,

```
if (şərt)
əməliyyat;
```

düzgün bir kod parçasıdır.Məsələn aşağıdakı proqrama baxaq:

```
using System;
class Program
{
   static void Main()
   {
      if (10 > 9)
         Console.WriteLine("10 > 9 sherti duzgundur");
      if (10 > 20)
         Console.WriteLine("Bu setir ekranda gorunmeyecek");
         Console.ReadKey();
   }
}
```

Burada if( 10 > 9) koduna fikir verək. 10 ədədi 9 ədədindən böyük olduğu üçün, 10 > 9 bərabərsizliyi düzgün bir şeydir, yəni 10 > 9 ifadəsinin nəticəsi doğrudur (true). Deməli, birinci if - ə aid kod icra olunacaq, ikinci if – in şərti düzgün olmadığı üçün isə, ikinci if - ə aid kod icra olunmayacaq. Beləliklə nəticə aşağıdakı kimi olur:




Burada qarşımıza yeni bir şey çıxdı: 10 > 9 ifadəsindəki “>” hissəsi. Bu bir müqaiyə operatorudur və vəzifəsi sağdakı ilə soldakını müqayisə etməkdir. Əgər, soldakı sağdakından kiçikdirsə, operator düzgün bir qiymət verir. Əks halda yalnış...

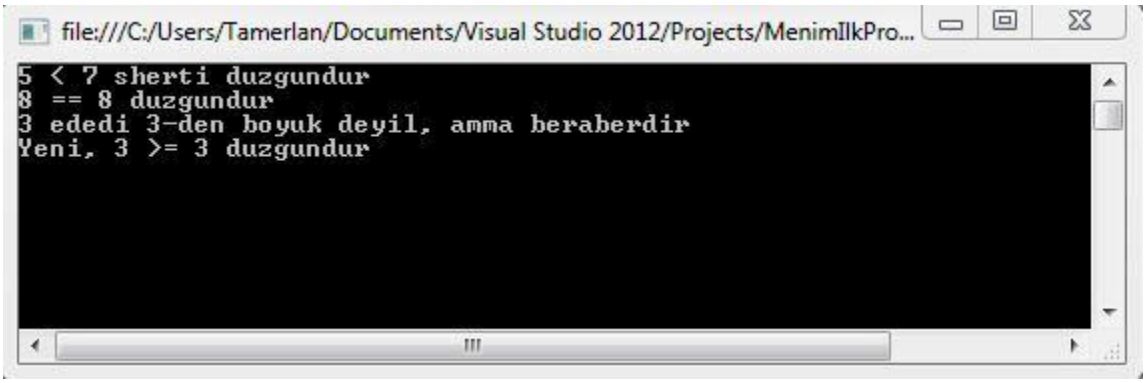
“>” ilə yanaşı digər digər müqayisə operatorları da var:

| | |
|---|---|
| > | böyükdür |
| < | kiçikdir |
| >= | böyükdür və ya bərabərdir |
| <= | kiçikdir və ya bərabərdir |
| == | bərabərdir |
| != | bərabər deyil |

Aşağıdakı misala baxaq:

```
using System;
class Program
{
    static void Main()
    {
        if (5 < 7)
            Console.WriteLine("5 < 7 sherti duzgundur");
        if (8 == 8)
            Console.WriteLine("8 == 8 duzgundur");
        if (5 != 5)
            Console.WriteLine("Bu setir ekranda gorunmeyecek");
        if (3 >= 3)
        {
            Console.WriteLine("3 ededi 3-den boyuk deyil, amma beraberdir");
            Console.WriteLine("Yeni, 3 >= 3 duzgundur");
        }
            Console.ReadKey();
    }
}
```

Nəticə aşağıdakı kimidir:




## for dövr ifadəsi

Proqramın axışını idarə edən növbəti kontrol ifadələrindən biri də dövr ifadələridir. Bəzən vəziyyət elə olur ki, poqramda bir əməliyyatı müəyyən sayda yerinə yetirmək – təkrar icra etmək lazım gəlir. Bunu etmək üçün qətiyyən ağlınıza, həmin təkrarlanacaq kodları yenidən alt-alta yazmaq üsulu gəlməsin. Bu çox mənasızdır. Bir fikirləşin, "salam" sözünü ekrana 50 dəfə yazmaq üçün, Console.WriteLine("salam"); sətrini 50 dəfə yazası deyilsiniz ki? Özü də bəzən təkrarlamaların sayı, əvvəlcədən məlum olmur. Bax bu kimi bir işi dövrə salaraq təkrarən yerinə yetirmək üçün dövr kontrol ifadələrindən istifadə olunur. Bu ifadələrdən biri də **for** – dur. for ifadəsinin ən çox istifadə olunan sintaktik şəkli aşağıdakı kimidir:

for(dövr dəyişəni; şərt; dövr dəyişənin artımı)

{

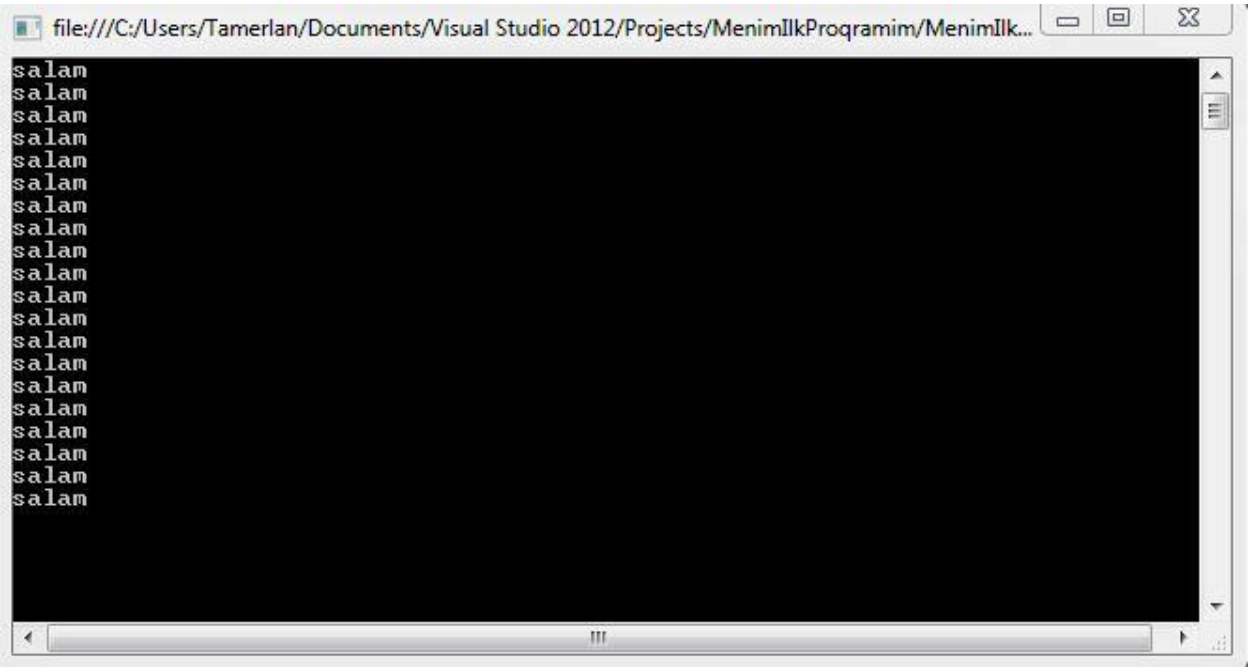
    Əməliyyatlar...

}

Burada **dövr dəyişəni** dövrü idarə edən dəyişəndir. **Şərt** hissəsi, dövrün şərtidir, yəni nə qədər ki bu şərt düzgündür, əməliyyatlar durmadan təkrar-təkrar icra olunur. Və dövr hər dəfə icra olunduqca dövrü idarə edən dəyişənin qiyməti, **dövr dəyişənin artımı** hissəsində qeyd olunduğu formada dəyişir. Məsələn, "salam" sözünü ekrana 20 dəfə çıxaraq:

```
using System;
class Program
{
    static void Main()
    {
        int i;
        for (i = 0; i < 20; i = i + 1)
            Console.WriteLine("salam");
            Console.ReadKey();
    }
}
```

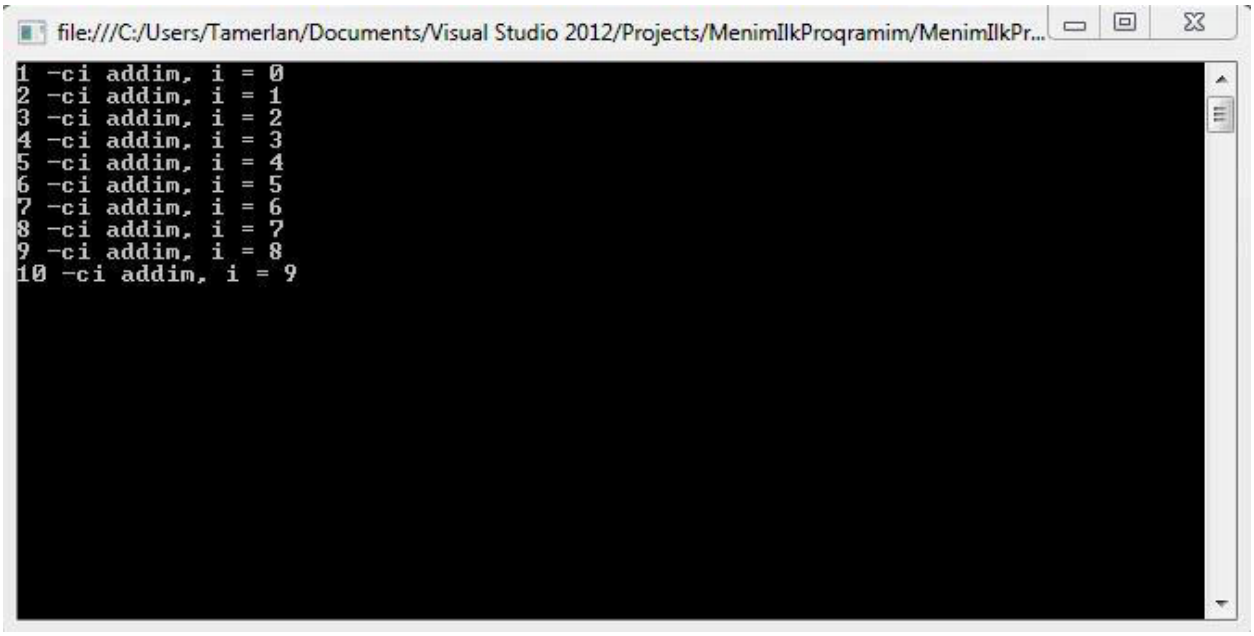
for (i = 0; i < 20; i++) hissəsinə fikir verin, deməli yuxarıda əvvəlcə bir dəyişən təyin etdik. Birinci hissəyə baxın, i dəyişəni dövrü idarə edən dəyişən olur, birinci hissədə bu dəyişənə 0 qiymətini verdik, i = 0. İkinci şərt hissəsinə baxın, nə qədər ki i < 20 şərti ödənir, dövrü yerinə yetir. Üçüncü hissədə isə i dəyişənin qiymətini bir-bir artırırıq. Bu artım və beləliklə dövr o zamana qədər təkrarlanır ki, i dəyişənin qiyməti 20 – ni aşsın. Çünki bu zaman dövrün şərti pozulur. Beləliklə, nəticə aşağıdakı kimi olur:




Növbəti misala baxaq:

```
using System;
class Program
{
    static void Main()
    {
        int i;
        for (i = 0; i < 10; i++)
            Console.WriteLine((i+1) + " -ci addim, i = " + i);
            Console.ReadKey();
    }
}
```

Aydındır ki, nəticə aşağıdakı kimi olacaq:




Dövr kontrol ifadələrini, o cümlədən, for, "proqram kontrol ifadələri" bölməsində, həmçinin operatorlar "Operatorlar" bölməsində geniş qeyd olunacaq. Hələki bəzi şeyləri başa düşmək üçün, qısa şəkildə qeyd olundu.

# BÖLMƏ 2. Verilənlər tipləri və Dəyişənlər

## Verilənlər tipi, tip nədir?

Proqramlaşdırma dilləri üçün **tip** vacib bir anlayışdır. Bu, xüsusilə də C# üçün belədir. Çünki, C# verilənlər tipinə tam bağlı bir dildir. Verilənlər tipi, bir dəyişənin nə kimi məumatları özündə saxlaya biləcəyini, həmin dəyişənin yaddaşdakı nüfuzunu və həmin dəyişən üzərində hansı əməliyyatları yerinə yetirə biləcəyimizi təyin edən bir məntiqi anlayışdır. Bu tərifi bir az açıqlayaq: Tutaq ki, tam (int) tipində bir dəyişən təyin etdik. Deməli bu dəyişən özündə tam ədədləri saxlaya biləcək, çünki onun tipi tamdır. Digər tərəfdən, tam ədədlər üzərində nə kimi əməlləri yerinə yetirə bilərik? Gəlin birlikdə sayaq: tam ədədləri toplaya bilərik, vura, bölə, çıxa bilərik, onlardan kök ala bilərik, qüvvətə yüksəldə bilərik və s. Digər tərəfdən, tam tipə aid bir dəyişən yaddaşda 4bayt = 32 bit yer ayırır. Bax bu sadalanan mühakimələr, yəni dəyişəni necə manipulyasiya edəcəyimiz və yaddaşın strukturunu həmin dəyişənin tipi müəyyən etdi. "Aaa…" dediyinizi eşidirəm sanki, bəli, tip dediyimiz şey məhz dəyişənlərin kimliyidir. Tam tipdə dəyişəndən kök ala bilərik, çünki onun tipi tamdır. Amma eyni əməliyyatı "Tamerlan" sətri üçün edə bilmərik, çünki onun tipi String-dir.

## Dəyişən anlayışı

Dəyişən anlayışını yuxarıda qeyd etmişdik. Bir daha təkrar edək. Dəyişən özündə hansısa bir məlumatı saxlayır. Həmin məlumat aydındır ki, yaddaşda (RAM - da) saxlanılır. Beləliklə, bir dəyişən yaddaşın bir hissəsidir. Məsələn, int a = 5; sətri yaddaşda 4 bayt həcmində adı "a" olan bir sahə ayırır və həmin yaddaş sahəsinə 5 informasiyasını yükləyir.

C# - da dəyişənlər iki qrupa bölünür:

-Lokal dəyişənlər

- Qlobal dəyişənlər

Qlobal dəyişənlər bir sinfə aid örnək (instance) dəyişənlərdir. Qlobal dəyişənləri yerləşdiyi sinfin hər bir nöqtəsindən müraciət etmək olur. Qlobal dəyişənlərə sahələr (fields) də deyilir. Qobal dəyişənləri "Siniflər" mövzusunda daha yaxşı başa düşəcəksiniz.

Lokal dəyişənlər, bir sinif daxilindəki hər hansısa kontekst içərisində təyin olunan və ancaq həmin kontekst (buna əhatə dairəsi də deyilir) içərisində müraciət edilə bilər dəyişənlərdir. Məsələn metodların parametrləri, metodların gövdələrində təyin olunan dəyişənlər lokal dəyişənlərdir. Bir lokal dəyişənə, bütün sinif daxilində müraciət etmək olmaz.

Bildiyimiz kimi bir dəyişənin tipi olur və tipin də nə olduğunu yuxarıda qeyd etdik. Bir dəyişəni aşağıdakı kimi bir deklorasiya (təyin) edirlər:

<**Tip**> <**dəyişənin adı**>;

Burada tip, dəyişənin tipidir. Məsələn, int, string, byte və s. String tipdə "str" adlı bir dəyişəni aşağıdakı kimi təyin edirlər:

**String** str;

Hələki dəyişənlərə bu nöqtədə son qoyaq və tiplərə davam edək. C# - da verilənlər tipləri 2 yerə ayrılır:

- Dəyər tipləri (valuable types)

- Referans tipləri (reference types)

Bu tiplər bir-birlərindən çox fərqlənir əslində. Dəyər tipləri, C# - ın standart tipləridir. Dəyər tiplərinə aid bir dəyişən, özünə mənimsədilən qiyməti birbaşa özündə saxlayır. Referans tipinə aid olan bir dəyişən isə, məlumatları birbaşa özündə saxlamır, bunun

yerinə həmin məmulatın saxanıldığı yaddaşın ünvanını özündə saxlayır, yəni həmin yaddaşa istinad edir. Buna görə onlara "refrans" tipli dəyişənlər deyilir. Referans tiplərini geniş şəkildə "Siniflər" bölməsində qeyd olunacaq. Hələki dəyər tipləri bizi maraqlandırır. C# - da dəyər tiplərinin özü 4 yerə bölünür:

1. Tam tiplər

2. Kərs tiplər

3. Simvol tiplər

4. Məntiqi tiplər

**Tam tiplər aşağıdakılardır:**

1. byte – 8 bit (1 bayt) həcmə maikdir [0-255] parçasında tam qiymət alır.

2. sbyte – 8 bit həcmə malikdir, [-128-127] parçasında tam qiymət alır.

3.ushort – işarəsiz (unsigned) tipdir, 16 bit həcmə malikdir və [0 - 65535] parçasında qiymət alır.

3. short – işrəlidir, 16 bit həcmi var, [-32,768 - 32,767] parçasında tam qiymət alır.

4. uint – işarəsiz (unsigned) tap tipdir, 32 bit həcmə malik [0 - 4,294,967,295] parçasında qiymət alır.

5. int – 32 tam həcmi var, [-2,147,483,648 -- 2,147,483,647] parçasında qiymət alır.

6. ulong – 64 bit işarəsiz və [0 - 18,446,744,073,709,551,615] parçasında qiymət alır.

7. long – 64 bit işarəli tipdir və [-923,372,036,854,775,808 - 9,223,372,036,854,775,807] parçasında qiymət alır.

**Kərs tiplər aşağıdakılardır:**

1. float - 32 bit həcmə malikdir. Vergüldən sonra 6 mərtəbə dəqiqliyinə malikdir. İşarəlidir, yəni həm mənfi həm də müsbət qiymətlər ala bilər.

2. double – 64 bit həcmə var, vergüldən sonra 14 mərtəbə dəqiqliyə malikdir. İşarəlidir.

3. decimal – 128 bit həcmi var, vergüldən sonra 27 mərtəbə dəqiqliyə malikdir və işarəlidir.

**Simvol tiplər aşağıdakılardır:**

1. char – bu tip özündə bir tək Unicode massivinə daxil olan simvolu saxlaya bilər və 16 bit həcmə malikdir.

**Məntiqi tiplər aşağıdakılardır:**

bool – bu tipə aid dəyişən sadəcə "true" və ya "false" (doğru və ya yanlış) qiymətlərindən birini ala bilər. Məntiqi əməliyyatlarda istifadə olunur, bir şərtin düzgün olub olmamağını müəyyənləşdirmək və s. Hallarda özünü göstərir.

Bir dəyişənə mənimsədilən qiymətlər, həmin dəyişənin tipi ilə uyumlu olmalıdır. Məsələn

```
int a = 45.5;
```

kod sətri düzgün deyil, çünki tam tipdə bir dəyişənə, tam olmayan bir qiymət verməyə çalışdıq. Eyniylə

```
bool a = 50;

char b = 45;

char c = "Salam";

string f = 456.85;
```

sətirlərinin hər biri səhvdir. Qeyd olunduğu kimi char simvol tipi, özündə tək bir simvolu saxlaya bilər və char dəyişənə mənimsədilən qiymət tək dırnaq işarəsi arasında yazılmalıdır. Məsələn

```
char a = 'A';

char b = 'b';
```

kod sətirlərinin hər biri düzgündür. Amma

```
char a = "A";

char b = 'bbb';
```

ifadələri düzgün deyil.

Bir dəyişənə, onun tuta biləcəyi qiymətdən böyük qiymət verməməliyik. Məsələn, int tipinin ala biləcəyi ən böyük müsbət qiymət 2,147,483,647 olduğundan aşağıdakı kod

sətri, bizə "Cannot implicitly convert type 'long' to 'int'. An explicit conversion exists (are you missing a cast?)" kimi bir xəta verəcək:

int a = 88444488444;

88444488444 ədədi int tipinin qiymətlər diapazonunu aşır, bu ədədi long tip özündə saxlaya bilər.

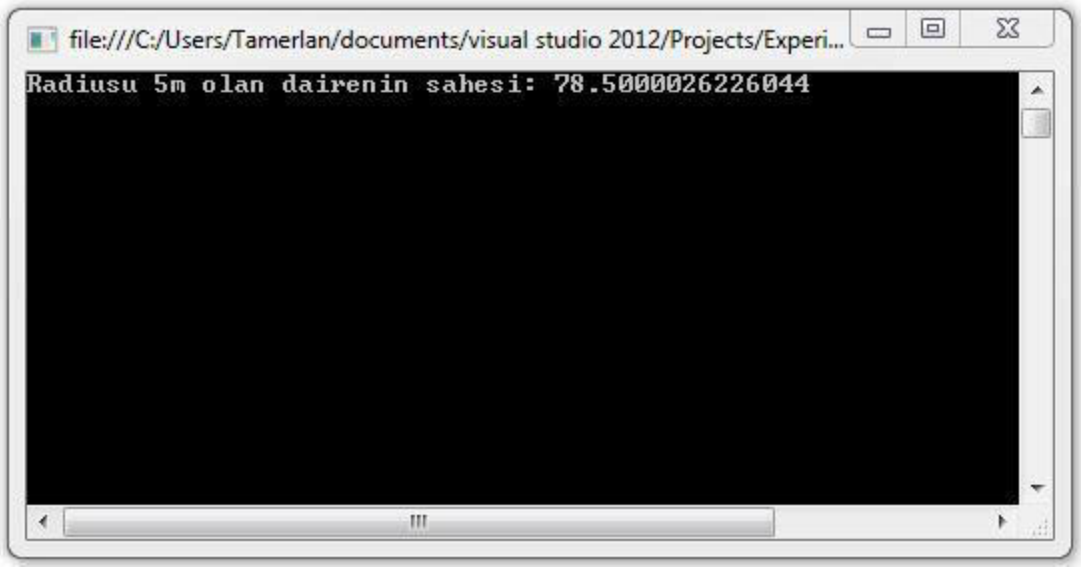
long a = 88444488444; tamamilə düzgündür.

Tam və kəsr tipləri əks etdirən bir proqram nümunəsinə baxaq:

Radiusu tam olan bir dairənin sahəsini hesablayan proqrama baxaq. Bildirimiz kimi, dairənin sahəsi, onun radiusununun kvadratının pi ədədinə hasilinə bərabərdir. Proqramımızı yazaq:

```
using System;
using System.IO;
class Soft
{
    public static void Main()
    {
        int radius = 5;
        float pi = 3.14f;                    // (*)
        double sahe = 0;
        sahe = radius * radius * pi;
        Console.WriteLine("Radiusu " + radius + "m olan dairenin sahesi: " + sahe);
        Console.ReadKey();
    }
}
```

Nəticə aşağıdakı kimi olacaq:




Beləliklə, radiusu 5m olan dairənin sahəsi təqribən 78.5 kvadrat metrdir.

Burada (*) sətrinə diqqət edin. Mənimsədilən qiymətin sonunda "f" simvolu qeyd olunub. Buna literal deyilir. Mənimsədilən qiymətin həqiqətən də float tipə uyğun olduğunu dəqiqləşdirmək üçündür. Double tipi üçün bu suffiks "d", decimal üçün isə "m" – dir.

## Mənimsətmələrə yaxından baxış

Bəlkə də heç bir proqramlaşdırma dilində mənimsətmə kimi vacib bir ikinci əməliyyat yoxdur. Bir dəyişənə bir qiymətin mənimsədilməsi, ən fundamental proqramlaşdırma prosesidir. Dəyişənlərə mənimsədilən qiymət onun tipi ilə uyğun olmalıdır. Tam tipdə bir dəyişən təyin edək və ona qiymət verək:

```
Int a;

a = 5;
```

C# - da mənimsətməni, dəyişənin təyin olunma sətrində də birbaşa yerinə yetirə bilərik. Yuxarıdakı iki kod sətrini bir sətirdə belə yaza bilərik:

```
Int a = 5;
```

Eyniylə,

```
Int a;

Int b;
```

Int c;

a = 10;

b = 15;

c = 76;

kodarını, aşağıdakı kimi tək bir sətirdə yaza bilərik;

int a = 10, b = 15, c = 76;

Gördüyünüz kimi, dəyişənlərə qiymət, təyin olunma sətrində dəyişənin adı qeyd olunduqdan sonra mənimsədildi. Bu nümunə həm də göstərir ki, eyni tipə malik bir neçə dəyişəni, bir-birindən vergüllə ayıraraq eyni bir sətirdə təyin etmək olar.

## Dəyişənlərin yaşama müddəti

Başa düşmək lazımdır ki, bir dəyişən, təyin olunduqdan sonra, həmişə mövcud olmur. Bütün insanların ömrü olduğu kimi, dəyişənlərin də ömrü var. Dəyişənlər, təyin olunduqları kontekst və ya skop daxilində mövcuddurlar. Məsələn aşağıdakı nümunəyə baxaq:

```
using System;
class Soft
{
   public static void Main()
   {
      for (int i = 0; i < 10; i++)
      {
         Console.WriteLine(i);
      }

      i = 50;
      /*Xəta var, i dəyişəninə ancaq
       * dövrün gövdəsində müraciət edilə bilər */
   }
}
```

Bu kod bizə “The name 'i' does not exist in the current context” xətasını verəcək. Çünki i dəyişəni, ancaq for dövrünün gövdəsi üçün keçərlidir. Dövrdən çıxdıqda artıq “i” adlı bir dəyişən olmayacaq. İndi aşağıdakı proqrama baxaq:

```
using System;
class Soft
```

```
{
    public static void Main()
    {
        int i;
        for (i = 0; i < 10; i++)
        {
            int a = 50;
            Console.WriteLine(i);
        }

        i = 50;
        a = 10; //xəta var
    }
}
```

Burada i adlı dəyişən, Main() metodunun gövdəsində təyin olunduğu üçün, həmin dəyişən, Main – in içərsində hər yerdə "yaşayır". a dəyişəninə fikir verək, bu dəyişən, for – un gövdəsi içərisində təyin olunub. Yəni bu dəyişən, ancaq for – un govdəsi üçün keçərlidir, dövrdən kənarda a = 10; ifadəsi xətaya səbəb olacaq. Bir də aşağıdakı nümunəyə baxaq:

```
using System;
class Soft
{
    public static void Main()
    {
        int i;
        for (int i  = 0; i < 10; i++)
        {
            int a = 50;
            Console.WriteLine(i);
        }
    }
}
```

Bu proqram bizə "A local variable named 'i' cannot be declared in this scope because it would give a different meaning to 'i', which is already used in a 'parent or current' scope to denote something else" xətasını verəcək. Deməli, i dəyişəni Main() metodu içərisində təyin olunduğu üçün, bu dəyişən Main() – in gövdəsi içərisində hər yerdə tanınır, o cümlədən for dövrü üçün də bu dəyişən tanınır. Təzədən for dövrü üçün "i" adlı dəyişən təyin etmək, bu dəyişənlə Main() içərisində təyin olunmuş eyni adlı dəyişənin toqquşmasına səbəb olacaq. Buna görə də dəyişənləri təyin edəndə, o dəyişənin hansı müddətdə lazım olacağını nəzərə almaq lazımdır.

# Tip Çevrilmələri və Tip Mənimsətmələr

### Avtomatik Tip çevrilmələri

Bir çox halarda bir tipə aid dəyişəni, başqa bir tipə aid olan dəyişənə mənimsətmə zərurəti yaranır. Məsələn aşağıdakı proqramda bir tam tipdə dəyişənə byte tipdə dəyişən mənimsədilib:

```
using System;
class Soft
{
   public static void Main()
   {
      int a;
      byte b = 20;
      a = b;
   }
}
```

Bu şəkildə mənimsətmə zamanı bir çevrilmə əməliyyatı həyata keçir. Yəni, byte→int çevrilməsi baş verir. byte tipi 8 bit, int 32 bit həcmə malikdir və ikisi də tam tipdir. Yəni, int tipi onsuz da byte tipini öz içərisində saxlayır. Yəni bu kimi mənimsətmədə heç bir problem yoxdur. Bax bu kimi mənimsətmələrdə həyat keçirilən çevrilmələrə avtomatik tip çevrilmələr deyilir. Çünki, bir byte dəyişənini int dəyişəninə mənimsətmək üçün əlavə bir şey etmədik. Beləliklə, sizin də təxmin edəcəyiniz kimi, avtomatik tip çevrilməsinin həyata keçməsi üçün aşağıdakı iki şərtin ödənilməsi zəruri və kafidir:

-Hər iki tip uyumlu olmalıdır

-Hədəf tip, dönüşdürülən tipdən böyük olmalıdır

Hədəf tip dönüşdürülən tipdən böyük olduğu üçün, dönüşdürülən tipə aid dəyişənin yaddaş sahəsi artır. Buna görə də avtomatik çevrilmələrə genişlədici çevrilmə (widening conversation) deyilir. Beləliklə, byte→int, int → long, uint→ulong, short→int kimi çevrilmələr, avtomatik həyata keçəcək. Buraya qədər hər şey aydındır elə deyilmi? Nə gözəl.

### Açıq tip mənimsətmələr

İndi maraqlı bir məqama toxunaq. Avtomatik tip çevrilmələrini başa düşdünüz, çətin bir şey yoxdur. Aşağıdakı proqrama bir baxın:

```
using System;
class Soft
{
```

```
    public static void Main()
    {
        int a = 100;
        byte b = a;
    }
}
```

Ağlınıza nə gəlir? Sizcə avtomatik çevrilmə olacaq? Bəli, düz fikirləşirsiz avtomatik tip çevrilməsi yerinə yetirilməyəcək. Çünki, avtomatik çevrilmə üçün zəruri olan ikinci şərt ödənmir. Amma belə baxanda byte tipində olan dəyişən [0 - 255] parçasında qiymət alırdı. Biz də b dəyişəninə tam tipdə də olsa, qiyməti 255-i aşmayan bir dəyişən mənimsətdik. Niyə də olmasın? Hər şeydən başqa byte 100 – ü özündə saxlayır. Amma bu tip mənimsətmələri bu formada aşkar (açıq) şəkildə yaza bilmərik. Bu yazılış bizə "Cannot implicitly convert type 'int' to 'byte'. An explicit conversion exists" xətasını verəcək. Belə çevrilmələr üçün açıq (aşkar) tip mənimsətmələrindən istifadə etmək lazımdır. Bu zaman mənimsətmə operatorunun sağ tərəfində mötərizələr içərisində hədəf tipin adı qeyd olunmalıdır. Yuxarıdakı proqamı aşağıdakı kimi yazaq:

```
using System;
class Soft
{
    public static void Main()
    {
        int a = 100;
        byte b = (byte) a;
    }
}
```

Beləiklə, xəta aradan qalxacaq. Sizin də təxmin etdiyiniz kimi, açıq tip mənimsətməni həyata keçirmək üçün aşağıdakı şərtin ödənməsi zəruri və kafidir:

-Hədəf tip, çevrilən tiplə uyğun olmalıdır

Odur ki, string → int kimi bir çevrilməni, açıq tip mənimsətmə ilə edə bilmərik.

Buraya qədər hər çey çox yaxşı, bir də aşağıdakı proqrama baxaq:

```
using System;
class Soft
{
    public static void Main()
    {
        int a = 300;
        byte b = (byte) a;
```

```
        Console.WriteLine(b);
    }
}
```

Aha! a dəyişənin qiymətinə fikir verin, 300 bayt tipinin qiymətlər diapazonunu aşır. Bu proqramı icra etsək, ekrana 44 çıxacaq. Bəs niyə 300 yox, 44? Gəlin araşdıraq:

Byte tipi 8 bit yer tutduğu üçün, 8 mərtəbəli bit sistemində (ikilik say sistemində) ifadə olunan ən böyük ədəd, aydındır ki, bütün bitlərin "1" olduğu haldır. Yəni

11111111 = 255

İndi, 300 ədədinin ikilik sistemdə təsvir edək:

100101100

İndi bu ikilik təsvir ilə 255- in iliklik təsvirini məntiqi VƏ (konyunksiya) əməliyyatından keçirək:

```
  011111111
& 100101100
-----------
  000101100 = 44
```

Maraqlıdır elə deyil mi? Məncə də çox maraqlıdır. Qeyd edim ki, mən sizin ikilik say sistemi ilə bağlı məlumatınızın olduğunu güman edirəm.

Qayıdaq əsas mövzuya, deməli 300 əvəzinə 44 çıxdı ekrana. Yəni, məlumat düzgün təsvir olunmadı – məlumat itkisi oldu. Buna görə də açıq tip mənimsətməni həyata keçirərkən, diqqət etmək lazımdır ki, hədəf tip mənimsədilən qiyməti özündə saxlaya bilsin. Çünki, əks hada məlmat itkisi (data lost) olacaq və proqramınız istədiyiniz kimi nəticə verməyə bilər.

## Manual Tip Çevrilmələri

Bir tipdə olan dəyişəni, başqa tipdə olan dəyişənə mənimsətmək, proqramlaşdırmada tez-tez istfadə olunur. Amma bəzən mənimsədilən dəyişənin tipi, hədəf tipdən fərqli olur. Məsələn, string → int çevrilməsi heç vaxt avtomatik yerinə yetirilə bilməz. Avtomatik çevrilmənin qaydalarını yadınıza salın, belə vəziyyətdə qaydaların heç biri ödənmir. Hətta açıq tip mənimsətmədən də istifadə edə bilmərik. Bəs, məsələn simvol olaraq sadəcə rəqəmlərdən ibarət olan bir string dəyişəni tam tipə necə çevirək? Niyə də olmasın? Məsələn,

String str = "4589";

dəyişənini bir tam tipdə olan dəyişənə mənimsətmək istəyə bilərik. Burada mənasız bir çey yoxdur. Nəticə etibarilə, str dəyişəni sadəcə rəqəmlərdən ibarətdir. Bu halda

int a = str; və ya int a = (int) str;

ifadələrinin hər biri xətaya səbəb olacaq. Deməli, alternativ çıxış yolu lazımdır. .NET Framework işərisində gələn Convert sinfi məhz bu iş üçün nəzərdə tutulub. Bu sinfin içərisində müxtəlif metodlar var ki, bu şəkildə uyumsuz tiplər arasında çevrilməni təmin edir (hələki sinif və metod anlayışınız yoxdur, narahat olmayın, rahat nəfəs alın və hələki müəyyən şeyləri əzbərləyin). Həmin metodlar aşağıdakılardır:

**ToByte(qiymət)** – parametrinə ötürülən qiyməti byte tipinə çevirir və geri qaytarır.

**ToInt16(qiymət)** – parametrinə ötürülən qiyməti short tipinə çevirir və geri qaytarır.

**ToInt32(qiymət)** – parametrinə ötürülən qiyməti int tipinə çevirir və geri qaytarır.

**ToInt64(qiymət)** – parametrinə ötürülən qiyməti long tipinə (yəni 64 bitlik int- ə) çevirir və geri qaytarır.

**ToDouble(qiymət)** – parametrinə ötürülən qiyməti double tipinə çevirir və geri qaytarır.

**ToUInit32(qiymət)** – parametrinə ötürülən qiyməti uint tipinə çevirir və geri qaytarır.

**ToString(qiymət)** – parametrinə ötürülən qiyməti string tipinə çevirir və geri qaytarır.

Buraya hamısını yazmadım. Digərlərini də adlarına görə müəyyənləşdirmək üçün, çox kiçik məntiq lazımdır. İndi bir nümunəyə baxaq:

```
using System;
class Soft
{
   public static void Main()
   {
      string str1 = "64";
      string str2 = "10";
      int eded = Convert.ToInt32(str1);       //(1)
      int quvvet = Convert.ToInt32(str2);     //(2)
      long netice = 1;
      for (int i = 0; i < quvvet; i++)
      {
         netice = netice * eded;
      }
      Console.WriteLine("64 ededinin 10-cu quvveti: " + netice);
```

```
        Console.ReadKey();
    }
}
```

Bu proqram eded adlı dəyişənin quvvet adlı qüvvətini hesablayır. A dədinin N – ci qüvvəti, a ədədini N dəfə öz-özünə vurmaq deməkdir. Məsələn, 10 ədədinin 4-cü qüvvəti 10^4 = 10*10*10*10 = 10000 edir. Buna görə də dövr qurduq və hər dövrdə eded dəyişənini özünə vurduq. Burada xüsusilə (1) və (2) sətirlərinə diqqət edin. str1 və str2 dəyişənlərinin hər biri string tipdədir. Ona görə str1-i tam ədədə çevirmək üçün

```
int eded =Convert.ToInt32(str1);
```

şəklində kod yazdıq. Bu kodun mənası belədir: "str1 dəyişəninin qiymətinin bir tam ədəd versiyasını əldə et və onu eded adlı dəyişənə mənimsət". (2) kod sətri də eyni cür işləyir. Beləliklə, proqramın nəticəsi aşağıdakı kimi olur:

```
64 ededinin 10-cu quvveti: 1152921504606846976
```

Bu nöqtəyə qədər ədədləri əvvəlcədən daxil edirdik. İndi proqram açılanda klaviaturadan daxi edilən məlumatlar üzərində işləyək. Məsələn klaviaturadan daxil edilən ədədin faktorialını hesablayan proqram yazaq. Klaviaturada olan hər şey bir simvol olduğu üçün, klaviaturadan daxil edilənlər əslində birbaşa string kimi qəbul olunur. Ona görə də klaviaturadan daxi edilən məlumatı əvvəlcə tama çevirməliyik. Klaviaturanı oxumaq üçün Console sinfindəki statik ReadLine() metodundan istifadə edilir. Bu metod klaviaturadan daxil olunan simvolları yan-yana düzür və "Enter" düyməsi basılanda yan-yana düzülmüş simvolları bütov string kimi geri qaytarır.

Bir ədədin faktorialı, 1-dən həmin ədədə qədər (həmin ədəd də daxil olmaqla) olan ədədlərin hasilinə bərabərdir. Məsələn 4! = 1 * 2 * 3 * 4 = 24 edir. Beləliklə, faktorial hesablayan proqrammımız aşağıdakı kimi olacaq:

```
using System;
class Soft
{
    public static void Main()
    {
        string klaviatura; // (1)
        Console.Write("Eded daxil edin: ");
        klaviatura = Console.ReadLine(); //(2)
        int eded = Convert.ToInt32(klaviatura); //(3)
        long netice = 1;
        for (int i= 1; i <= eded; i++)
        {
```

```
            netice = netice * i;
        }
        Console.WriteLine(eded + "! = " + netice);
        Console.ReadKey();
    }
}
```

Bu proqramda (1) sətri bir string dıyişən təyin edir, bu dəyişən klaviturадan daxi olunanları özündə saxlayacaq. (2) sətri bu dıyişənə klaviaturadan daxil olunanları mənimsədir. (3) sətri bu dəyişənin və deməli klaviaturadan daxil olunanların qiymətini tam ədəd kimi eded dəyişəninə mənimsədir. Nəhayət faktorial hesablanır və ekran nəticəsi aşağıdakı kimi olur:

```
Eded daxil edin: 5

5! = 120
```

**QEYD:** 0! = 1;

Diqqətli olmağımız gərəkən bəzi nüanslar var. Deməli Məsələn ToInt32 metodu hansısa dəyişəni "zorla" tam ədədə çevirir. Yuxarıda string dəyişənlərin qiymətləri ancaq rəqəmlərdən ibarət idi və bu rəqəmlərin əmələ gətirdiyi tam ədəd, int tipinin qiymətlər diapazonunu aşmır. Bununla belə aşağıdakı proqrama baxaq:

```
using System;
class Soft
{
    public static void Main()
    {
        string s1 = "Laliko"; //(1)
        string s2 = "878964568415894561986451845"; //(2)
        int eded = Convert.ToInt32(s1);
        eded = Convert.ToInt32(s2);
        Console.ReadKey();
    }
}
```

Burada (1) sətrinə fikir verək. "Laliko" ifadəsi, bir tam ədəd kimi necə təsvir olunsun axı? Bu sətir bizə "FormatException" xətasını verəcək. Yəni format düzgün deyil. (2) sətrində isə str2 dəyişəninin qiyməti ancaq rəqəmlərdən ibarət olsa da, onun qiyməti int tipinin diapazonunu aşır. Bu isə bizə "OverflowException" xətasını verəcək.

## İdentifikatorlar

İdentifiktor proqram yazarkən, bir sinfin üzvlərinə və dəyişənlərə verdiyimiz addır. İdentifikator, proqramdakı obyektlərin kimliyidir. Məsələn

Int a = 50;

Ifadəsinə "a" bir identifikatordur. Yəni, o bir dıyişənin adıdır. Aşağıdakı kiçik proqrama baxaq:

```
using System;
class Soft
{
   public static void Main()
   {
      int a = 50;
      char l = 'L';
      Console.ReadKey();
   }
}
```

Bu proqramda 6 ədəd identifikator var. Gəlin birlikdə sayaq:

1. Sinfimizin adı: "Soft"

2. Başlanğıc metodun adı: "Main"

3. Tam dəyişənin adı: "a"

4. char dəyişənin adı: "l

5. System kitabxanasındakı sinfin adı: "Console"

6. Console sinfindəki metodun adı: "ReadKey"

## Bəzi İpucları və Qaydalar

Bəlkə də bunu yazmaq gecdir, amma sistematik gedişatı pozmamaq üçün indiyə saxladım. Unutmamaq lazımdır ki, C# - da hər bir dəyişən istifadə olunmamışdan əvvəl təyin olunmalıdır. Bir dəyişən təyin etməmiş onu istifadə emək olmaz. Bir dəyişəni istifadə etməzdən əvvəl onun hökmən bir qiyməti olmalıdır. Yoxsa "Use of unassigned local variable 'ad'" xətasını alacağıq. Eyni bir əhatə dairəsində eyni adlı iki dəyişən ola bilməz. Dəyişənlərə ad verərkən aşağıdakı qaydaları gözləmək, daha professional kod yazılışına uyğundur:

1. İki dəyişənin adı bir-birindən sadəcə böyül-kiçik hərf fərqi ilə fərqlənməsin. Bu əslində səhv deyil (C# dilinin CaseSensitive dəstəkli olduğunu yadınıza salın). Sadəcə qarışıqlığa səbəb olur. Məsələn

Int a;

Int A;

kimi...

2. Bir dəyişənin adı rəqəmlə başlaya bilməz, amma hərflə başlayıb ortada rəqəmlər iştirak edə bilər. Məsələn:

Int 5 – Səhv

Int 52a – səhv

Int a5 – düz

Int T4L – düz

3. Bir dəyişənin adında "%, $, #, &" kimi xüsusi simvollar ümumiyyətlə iştikar edə bilməz.

4. Bir dəyişənin adı C#  - ın açar sözləri ilə üst-üstə düşə bilməz. Məsələn

Int if = 5;

If C# - da şərt ifadəsini xarakterizə edir. Əgər hökmən identifikatorunuz bir xüsusi sözlə üst-üstə düşməlidirsə onda identifikatordan əvvəl "@" simvolunu əlavə etmək kifayətdir. Məsələn

Int @if = 5;

Console.WriteLine(@if);

5.Bir dəyişənin adında xüsusi simvollardan ancaq alt-tire (_) yazılmağına icazə verilir. Bu da əgər dəyişənin adı iki sözlə ifadə olunubsa bu sozləri bir-birlərindən ayırmaq üçün nəzərdə tutulub. Məsələn

Int kitab_sayı;

6. Bir dəyişən iki sözdən ibarətdirsə, onda birinci və ikinci sözlərin baş hərflərini böyük hərəflə, qalanlarını kiçik hərflə yazmaq daha gözəl görünüşə səbəb olacaq. Bu deklorativ qayda "Hungarian note" qaydasına uyğundur. C# - ın özü bu qaydadan istifadə edir.

# BÖLMƏ 3. OPERATORLAR

## Operator nədir?

Operatorlar, verilənlər üzərində işləmək, onların qiymətlərini dəyişmək, onlar üzərində riyazi əməliyyatlar aparmaq üçün istifadə olunan qaydalardır. Məsələn iki dəyişənin qiymətinin toplanması, bir dəyişənin qiymətini əksinə dəyişmə, bir dəyişənlə digər dəyişənin qiymətini müqayisə etmək və s. kimi əməliyyatar operatorlar vasitəsilə həyata keçirilir. Operatorun üzərində işlədiyi dəyişənə operand deyilir. Bütün proqramaşdırma dillərində operatorlar mövcuddur. C# proqramlaşdırma dilində operatorlar öz funksionallığına görə aşağıdakı kimi qruplaşdırıla bilər:

1. Vahid operatorlar (unary operators)

2. Cəbri operatorlar (arithmetic operators)

3. Müqayisə operatorları (relational operators)

4. Məntiqi operatorlar (logical operators)

5. Mənimsətmə operatorları (assignment operators)

6. Bit əsaslı operatorlar (bitwise operators)

7. Digər operatorlar

## Vahid operatorlar

C# - da vahid operatorlar, tək bir operand tələb edən, yəni tək bir operand üzərində işləyən operatorlardır və onlar aşağıdakılardır:

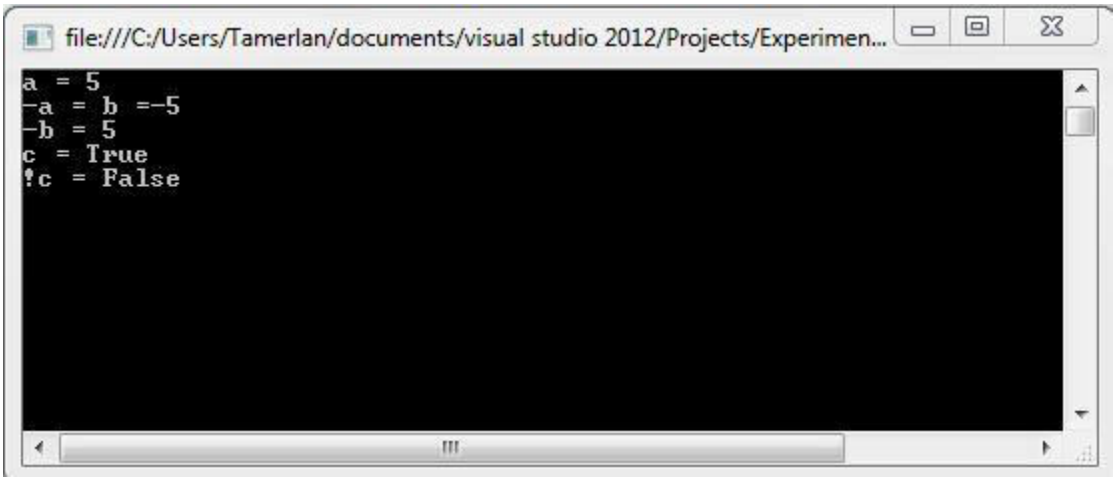
| Operator | Əməliyyat |
|---|---|
| ++ | Bir dəyişənin qiymətinin müsbət versiyasını verir |
| - | Bir dəyişənin qiymətinin mənfi versiyasını verir |
| ! | Bu operator əslində məntiqi operatordur və bir bool dəyişənin qiymətini əksinə çevirir. Yəni, true isə false, false isə true olur. |
| ++ | Bir dəyişənin qiymətini bir vahid artırır |
| -- | Bir dəyişənin qiymətini bir vahid azaldır |

Yuxarıda + operatorunun açıqlaması sizi çaşdırmasın. "Bir dəyişəninqiymətinin müsbət versiyasını verir" demək o demək deyil ki, mənfi bir dəyişənin qarşısına + yazanda əldə olunan nəticə onun müsbət variantı olur.

Aşağıdakı proqrama baxaq:

```
using System;
class Soft
{
   public static void Main()
   {
      int a = 5;
      int b = -a;
      Console.WriteLine("a = " + a);
      Console.WriteLine("-a = b =" + b);
      b = -b;
      Console.WriteLine("-b = " + b);
      bool c = true;
      Console.WriteLine("c = " + c);
      c = !c;
      Console.WriteLine("!c = " + c);
      Console.ReadKey();
   }
}
```

Aydındır ki, bu proqramın nəticəsi aşaıdakı kimi olur:




İndi isə çox maraqlı operatorlar olan ++ və -- operatorlarına baxaq. Qeyd olunduğu kimi

Bu operatorlar uyğun olaraq bir dəyişənin qiymətini bir vahid artırır və bir vahid azaldır. Məsələn,

x =x +1;

ifadəsini aşağıdakı kimi yaza bilərik:

x++;

Eyniylə də

x = x – 1;

sətrini

x--;

kimi də yaza bilərik. Bu operatorların hər biri operandın həm əvvəlində, həm də sonunda yazıla bilər. Məsələn

x = x +1;

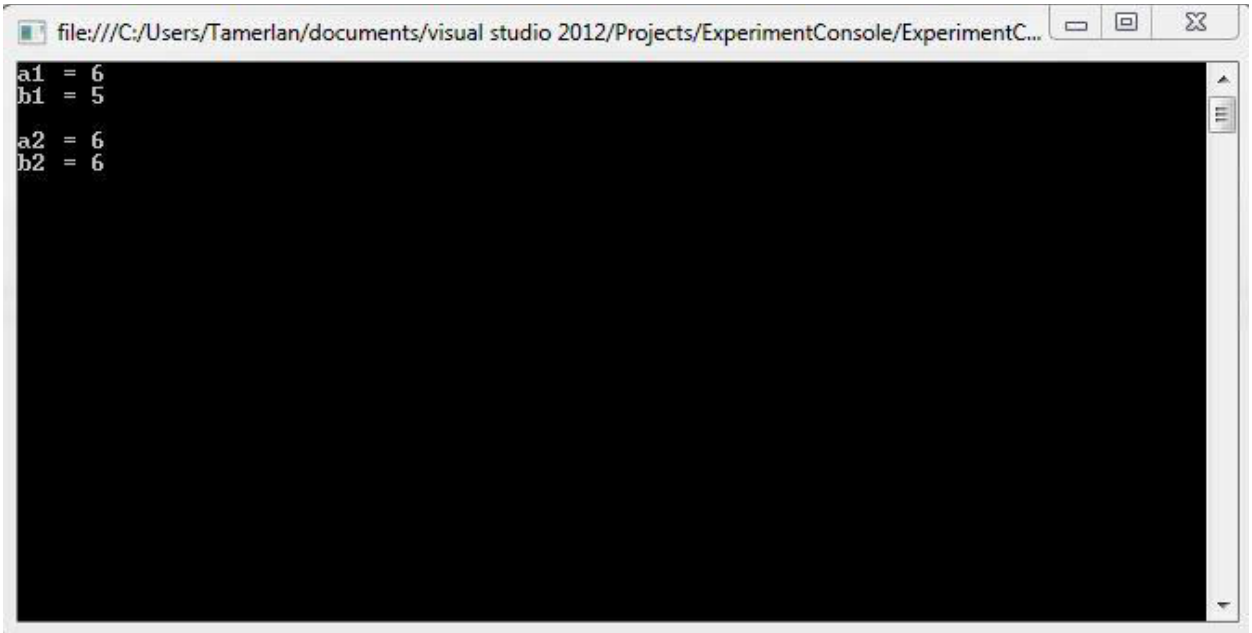
ifadəsini ++x;

kimi də yaza bilərik. Artırma və azaltma operatorlarının operandın əvvəlində və ya sonunda gəlməsi arasında heç bir fərq yoxdur. Yəni ++x və x++ ifadələrinin ər biri x dəyişəninin qiymətini bir vahid artırır. Amma daha mürəkkəb ifadələrdə bu operatorların yerləşmə mövqeyi fərqli nəticələrə səsəs ola bilər. Məsələn başqa bir dəyişənə x dəyişənin qiymətinin bir vahid çox vəziyyətini mənimsətmək istəyiriksə, onda ++ operatorunun əvvəldə və ya axırda yazılmağının fərqi var. Belə ki, əgər bu operatorlar mənimsətmə operatorunun yanında gələrsə, onda ++ (--) operatorlarına aid operandın qiyməti bir vahid artır (azalır) sonra yeni qiymət, müvafiq dəyişənə mənimsədilir. Əgər bu operatorlar mənimsətmə operatorunun yanında gəlməzsə, yəni operandın sağında yerləşərsə, onda ++ və ya -- operatorlarına aid operandın qiyməti müvafiq dyişənə mənimsədilir, sonra bir vahid artırılır və yaz azaldılır. Bu faktı aşağıdakı proqram göstərir:

```
using System;
class Soft
{
   public static void Main()
   {
      int a1 = 5;
      int b1 = a1++; //Operator sağda yerləşir
      Console.WriteLine("a1 = " + a1);
      Console.WriteLine("b1 = " + b1);
      Console.WriteLine();
```

```
        int a2 = 5;
        int b2 = ++a2; //operator solda yerləşir
        Console.WriteLine("a2 = " + a2);
        Console.WriteLine("b2 = " + b2);
        Console.ReadKey();
    }
}
```

Bu proqramın nəticəsi aşağıdakı kimi olur:




## Cəbri operatorlar

C# - da cəbri operatorlar aşağıdakılardır:

+ Toplama

- Çıxma

\ Bölmə

* Vurma

% Modul (qalıq)

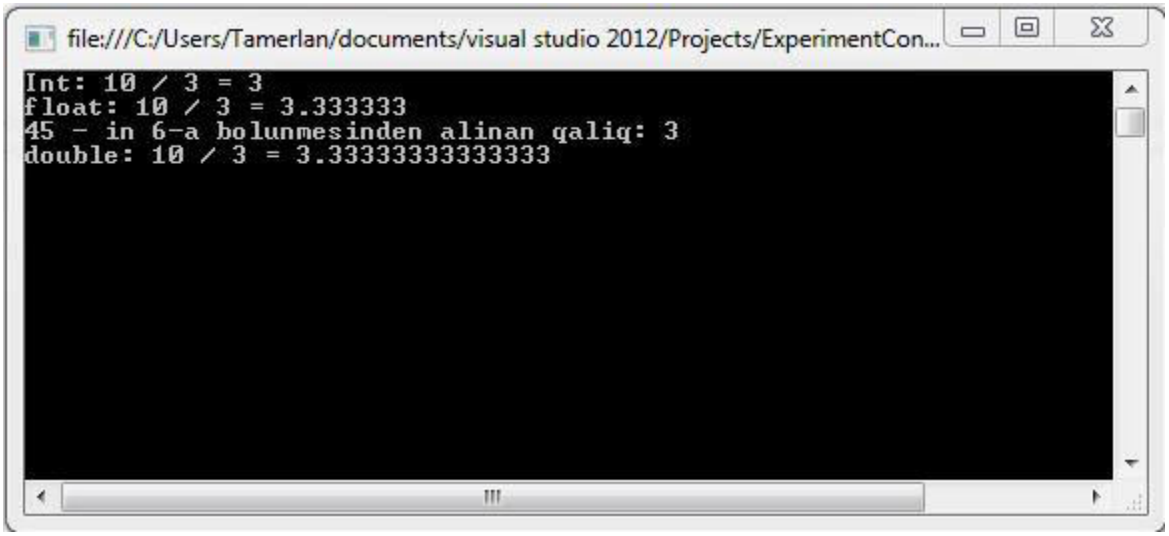
Bu operatorlar riyaziyyatda hansı əməliyyatı yerinə yetirirsə, C# -da da eyni əməliyyatı yerinə yetirir. Amma C# tip sisteminə dayalı bir dil olduğu üçün, bu operatorların təbiqinin nəticəsi, operandların tipinin nə olmasından asılı olaraq dəyişir. Məsələn

int a = 10 / 3;

sətrinə baxaq. Normalda 10 / 3 sonsuz dövru onluq kəsrdir və onun nəticəsi 3.3333... olur. Amma 10 / 3 –dən alınan nəticəni bir tam dəyişəmə mənimsətdiyimizə fikir verin. Bir tam dəyişən özündə kəsr ədədlər saxlayan bilmədiyi üçün burada 10 / 3 = 3 olacaq. Yəni nəticənin tam hissəsi qalacaq. Bir ədədin digərinə bölünməsinə alınan qalığı % operatorunun vasitəsi ilə əldə oluna bilər. Məsələn 10 % 3 = 1 edir.

Aşağıdakı proqrama baxaq:

```
using System;
class Soft
{
   public static void Main()
   {
      int a = 10 / 3;
      float b = 10 / 3f;
      int c = 45 % 6;
      double d = 10 / 3d;
      Console.WriteLine("Int: 10 / 3 = " + a);
      Console.WriteLine("float: 10 / 3 = " + b);
      Console.WriteLine("45 - in 6-a bolunmesinden alinan qaliq: " + c);
      Console.WriteLine("double: 10 / 3 = " + d);
      Console.ReadKey();
   }
}
```

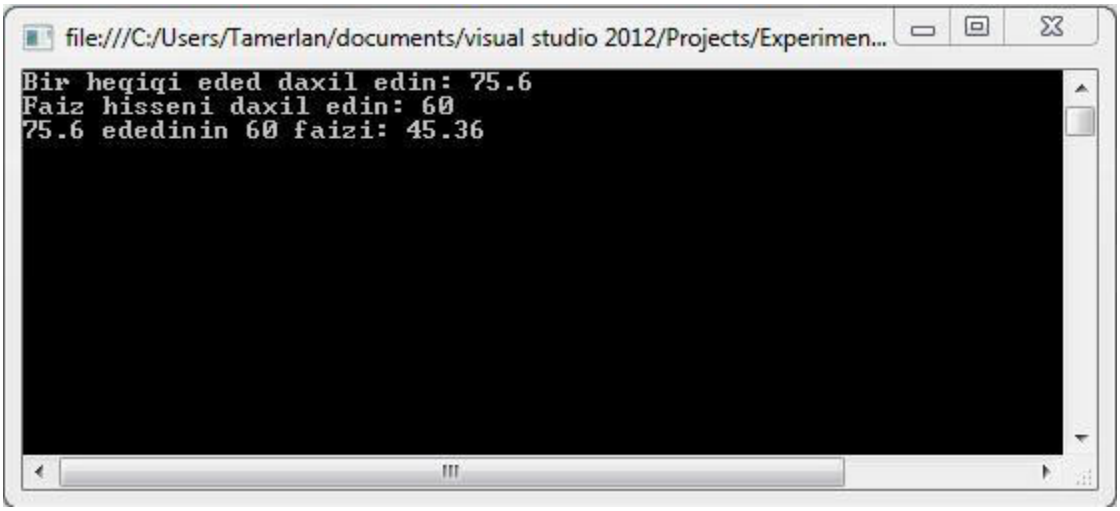



Burada b və d dəyişənlərinə eyni əməliyyatın nətcəsi mənimsədildi, bu tiplər kəsr tiplərdir. "Verilənlər tipləri" mövzusunda qeyd olunduğu kimi bu tiplər bir-birlərindən verguldən sonrakı dəqiqliyə görə fərqlənir.

İndi isə daxil edilən ədədin daxil edilən faizini tapan proqram yazaq, məlumdur ki, bir ədədin x faizi, onun x/100 hissəsinə bərabərdir:

```
using System;
class Soft
{
   public static void Main()
   {
      Console.Write("Bir heqiqi eded daxil edin: ");
      float eded = Convert.ToSingle(Console.ReadLine());     //(1)
      Console.Write("Faiz hisseni daxil edin: ");
      int faiz = Convert.ToInt32(Console.ReadLine());
      float netice = (eded * faiz) / 100;
      Console.WriteLine(eded + " ededinin " + faiz + " faizi: " + netice);
      Console.ReadKey();
   }
}
```

Bu proqramın nəticəsi aşağıdakı kimi our:




Bu proqramda çətin bir şey yoxdur, hər şey aydındır. Ola bilsin ki, (1) sətri sizə qəribə gəlsin. "Tip çevrilmələri" mövzusunu yadınıza salın. Klaviaturadan daxi ediləni bir ədəd kimi əldə etmək üçün əvvəlcə onu hansısa bir ədəd tipinə çevirməli idik. Bizim eded dəyişəni float tipdə olduğu üçün klaviaturanı float- a çevirdik. "Bəs float- a çevirdiksə, Convert.ToSingle() niyə yazdıq?" sualını verə bilərsiniz. Əslində, int, float, byte və s. bunların hamısı C# - ın açar sözləridir, yəni .NET Framework baza sinif kitabxanasına daxil olan struktların C#- dakı ləqəbidir. Odur ki, C# -dakı float, .NET Framework – da Single struktuna uyğundur, o cümlədən C# - ın int tipi, .NET Framework – da Int32 struktuna uyğundur.

## Müqayisə Operatorları

Müqayisə operatorları iki ədəd tipinə malik dəyişənin qiymətini müqayisə etməyə imkan verir. Əgər müqayisə ifadəsi bir münasibət kimi doğrudursa, bu operatorun nəticəsi true, əks halda false olur. C# - da müqayisə operatorları aşağıdakılardır:

| Operator | Mənası |
|---|---|
| == | Bərabərdir |
| != | Fərqlidir (bərabər deyil) |
| > | Böyükdür |
| < | Kiçikir |
| >= | Böyükdür və ya bərabərdir |
| <= | Kiçikdir və ya bərabərdir |

== operatoru iki dəyişənin qiymətinin bir-birinə bərabər olmasını yoxlaylır. Əgər qiymətlər bərabərdirsə, münasibətin nəticəsi true olur, əks halda false olur. != operatoru == operatorundan fərqli olaraq bir dəyişənin qiyməti digərinin qiymətinə bərabər deyilsə true qaytarır, əks halda false. Beləliklə == operatorunun true olduğu münasibətlərdə != operatoru false qiymətə malik olur və tərsinə. > operatoru əgər soldakı dəyişənin qiyməti sağdakının qiymətindən böyükdürsə və sadəcə böyükdürsə true qaytarır. Bərabərlik halı true nəticəyə kifayıt deyil. < operatoru da bu işin tərsini yerinə yetirir. >= operatoru əgər soldakının qiyməti sağdakının qiymətindən kiçik deyilsə true qiyməti qaytarır. Yəni operandların qiymətlərinin bir-birlərinə bərabər olması, true nəticəyə səbəb olur. <= operatoru da bu işin tərsini yerinə yetirir. Beləliklə, >= və <= operatorlarının iştirak etdiyi ifadənin nəticəsi o zaman true olur ki, operandların qiyməti bir-birlərinə bərabər olsun. Aşağıdakı nümunəyə baxaq:

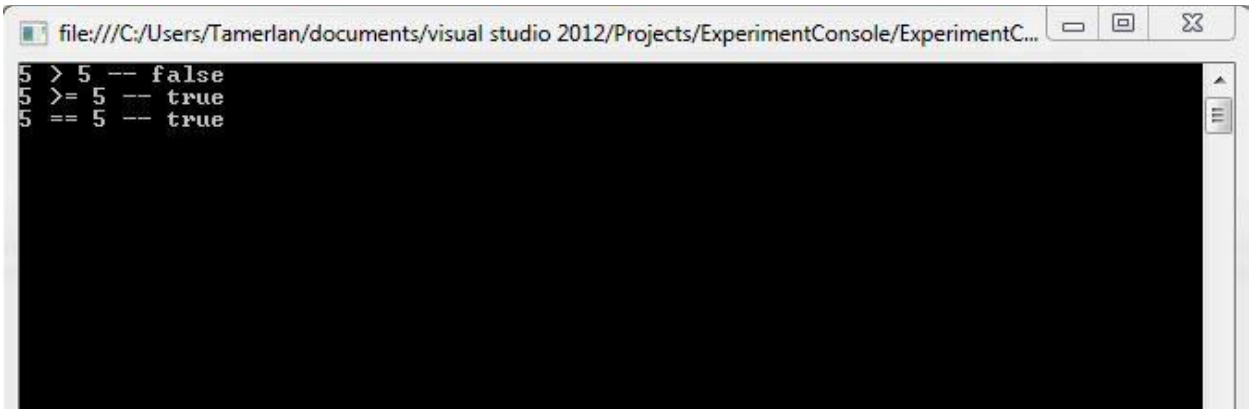
```
using System;
class Soft
{
   public static void Main()
   {
      if (5 > 5) Console.WriteLine("5 > 5 -- true");
      else Console.WriteLine("5 > 5 -- false");

      if (5 >= 5) Console.WriteLine("5 >= 5 -- true");
```

```
        else Console.WriteLine("5 >= 5 -- false");

        if (5 == 5) Console.WriteLine("5 == 5 -- true");
        if (5 != 5) Console.WriteLine("5 != 5 -- true");
        Console.ReadKey();
    }
}
```

Proqramın nəticəsi, aşağıdakı kimi olur:




Məsələn,

```
if (5 > 5) Console.WriteLine("5 > 5 -- true");
```

sətrində 5 > 5 səhv olduğu üçün if ifadəsinin şərti ödənmir və ona aid sətir icra olunmur. Digərləri də müvafiq qayda ilə.

## Məntiqi Operatorlar

Məntiqi operatorlar bool tipli dəyişənlər üzərində işləmək və klassik məntiq əməliyyatlarını yerinə yetirmək üçündür. Məsələn "mətbəxə gedib mənə bir bıçaq VƏ bir çəngəl gətirin" cümlələrində klassik məntiq (VƏ) özünü göstərir. Bu məntiqə əsasən mətbəxdən bıçaq tapmasanız mənə çəngəl gətirməməlisiniz ya da çəngəl olmasa mənim üstümə sadəcə bıçaqla gəlməməlisiniz. Əgər ikisi də varsa onda ikisini də gətirməlisiniz. Əgər mən cümləni "mətbəxə gedib mənə bir bıçaq VƏ YA bir çəngəl gətirin" şəklində ifadə etmiş olsaydım, onda mətbəxdə bıçaq yoxdursa mənə çəngəli, əgər çəngəl yoxdursa onda bıçağı, əgər hər ikisi varsa hər ikisini gətirməli idiniz. Məntiqi operatorlar da bu məntiqlə işləyir. Əgər bir azca diskret riyaziyyat anlayışınız varsa, onda məntiqi operatorları siz artıq bildiniz. C# -da məntiqi operatorlar aşağıdakılardır:

| Operator | Mənası | Diskret riyaziyyatda |
|---|---|---|
| & | VƏ | Konyunksiya |

| | | |
|---|---|---|
| \| | VƏ YA | Dizyunksiya |
| ^ | XOR | mod(2)-ə görə cəm |
| ! | İnkar | İnkar |
| && | Şərtə bağlı VƏ | |
| \|\| | Şərtə bağlı VƏ YA | |

& məntiqi operatorunun operandları qeyd olunduğu kimi, bir bool tipdə dəyişənlər və ya sabitlər olmalıdır. Bu operator yalnız o zaman true qiymət qaytarır ki, operandların hər ikisi true olsun. | operatoru o zaman yanlış qiymət qaytarır ki, operandların hər ikisi yanlış olsun. ^ operatoru o zaman doğru qiymət qaytarır ki, operantlardan biri false olduqda digəri true olsun. ! operatorunu vahid operatorlar hissəsində qeyd etmişdik. Bu operator tək bir operand tələb edən məntiqi operatordur. Operatorun məntiqi dəyişənə tətbiqinin nəticəsi, həmin dəyişənin əks qiymətinin əldə olunmasına səbəb olur. Yəni

!true = false və

!false = true

Diskret riyaziyyatda true və false yerinə 1 və 0 – dan istifadə olunur. Odur ki, true = 1, false = 0 qəbul olunur. Məntiqi operatorların operandları bool tipində olduğu kimi, bu operatorların da nəticəsi bool tipindədir. Aşağıdakı cədvər məntiqi operatorları və onların müvafiq məntiqi dəyişənlərə tətbiqinin nəticəsini əks etdirir:

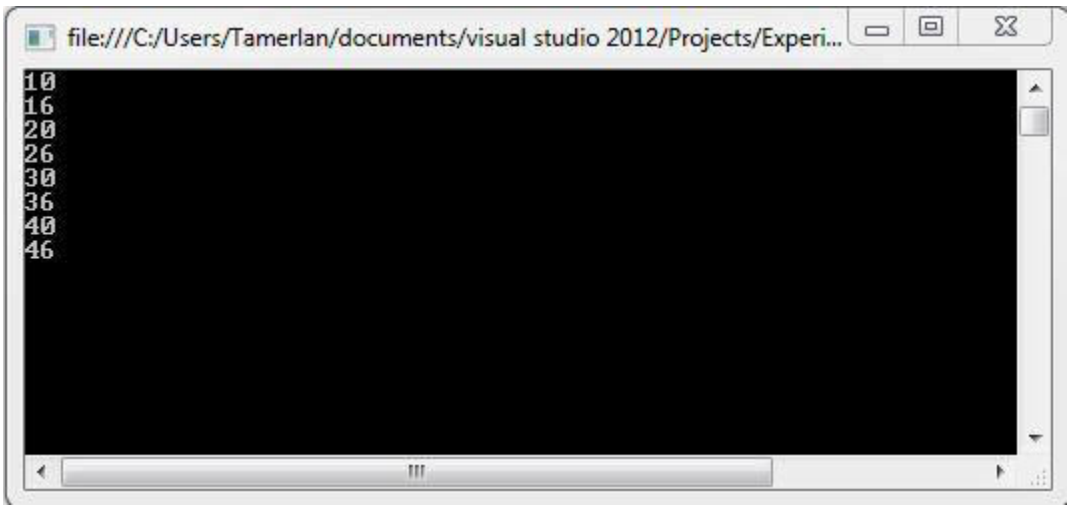
| a | b | a & b | a \| b | a ^ b | !a |
|---|---|---|---|---|---|
| true | true | true | true | false | false |
| true | false | false | true | true | false |
| false | true | false | true | true | true |
| false | false | false | false | false | true |

Aşağıdakı proqram baxaq. Bu proqram 10 ilə 50 arasındakı, sonuncu rəqəmi 0 ilə 6 olan cüt ədədləri əks etdirəcək. Məlumdur ki, cüt ədət 2-ə tam bölünən ədədlərdir.

```
using System;
class Soft
{
```

```
    public static void Main()
    {
        for (int i = 10; i < 50; i++)
        {
            if((i % 2 == 0) & (i % 10 == 0 | i % 10 == 6))
                Console.WriteLine(i);
        }
        Console.ReadKey();
    }
}
```

Proqramın nəticəsi aşağıdakı kimi olur:




If ifadəsinin içərisinə fikir verin.

(i % 2 == 0) & (i % 10 == 0 | i % 10 == 6))

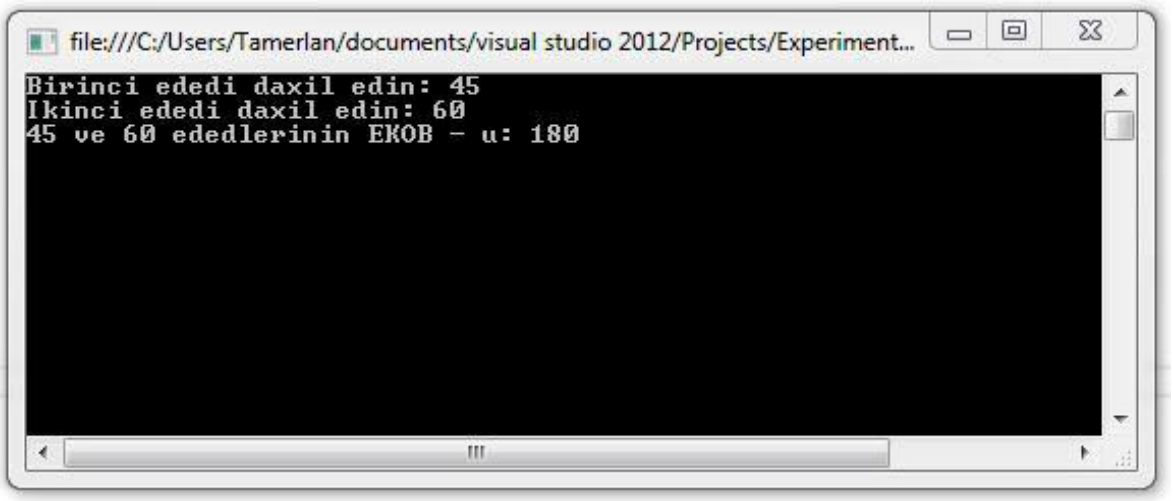
Ifadəsinin mənası o deməkdir ki, "2 - ə tam bolünən eyni zamanda (VƏ) sonuncu rəqəmi 0 VƏ YA 6 olan ədədləri ekranda çap et".

Növbəti misala baxaq. Bu proqram daxil edilən iki tam ədədin ƏKOB – nu tapacaq. a və b ədədlərinin ƏKOB – u elə c ədədinə deyilir ki, bu ədəd a və b ədədlərinin hər ikisiniə tam bölünən ədədlərin ən kiçiyi olsun.

```
using System;
class Soft
{
    public static void Main()
    {
        Console.Write("Birinci ededi daxil edin: ");
        int eded1 = Convert.ToInt32(Console.ReadLine());
```

```
        Console.Write("Ikinci ededi daxil edin: ");
        int eded2 = Convert.ToInt32(Console.ReadLine());
        int max;
        if (eded1 >= eded2)
            max = eded1;
        else
            max = eded2;
        bool saxla = true;
        for (int i = max; i < 1000000; i++)
        {
            if (saxla == true)
            {
                if ((i % eded1 == 0) & (i % eded2 == 0))
                {
                    Console.WriteLine(eded1 + " ve " + eded2 + " ededlerinin EKOB - u: " +
i);
                    saxla = false;
                }
            }
        }
        Console.ReadKey();
    }
}
```

Məlumdur ki, iki ədədin ƏKOB – u o ədədlərin böyük olanından kiçik ola bilməz. Çünki kiçik olsa, artıq həmin ədəd böyük olan ədədə tam bölünməz. Buna görə də, əvvəlcə daxil edilən iki ədədin ən böyüyünü max adlı dəyişənə mənimsətdik. Sonra bu max ədədindən 1000000 – a qədər dövr qurduq və həm eded1-ə həm də eded2-ə tam bölünən ə ədi ekranda əks etdirdik. Aydındır ki, bu ədəd ən kiçik bölünən ədəd olacaq. Çünki biz aşağıdan yuxarıya doğru gedirik. İlk bölünmədə artıq digər elementlərin yoxlanılmağı bizi maraqlandırmadığı üçün dövrü saxlamalıyıq. Bunun üçün də əlavə bir saxla adlı bool dəyişəndən istifadə etdik. Beləliklə proqramın nəticəsi aşağıdakı kimi olacaq:




## Şərtə Bağlı Məntiqi Operatorlar

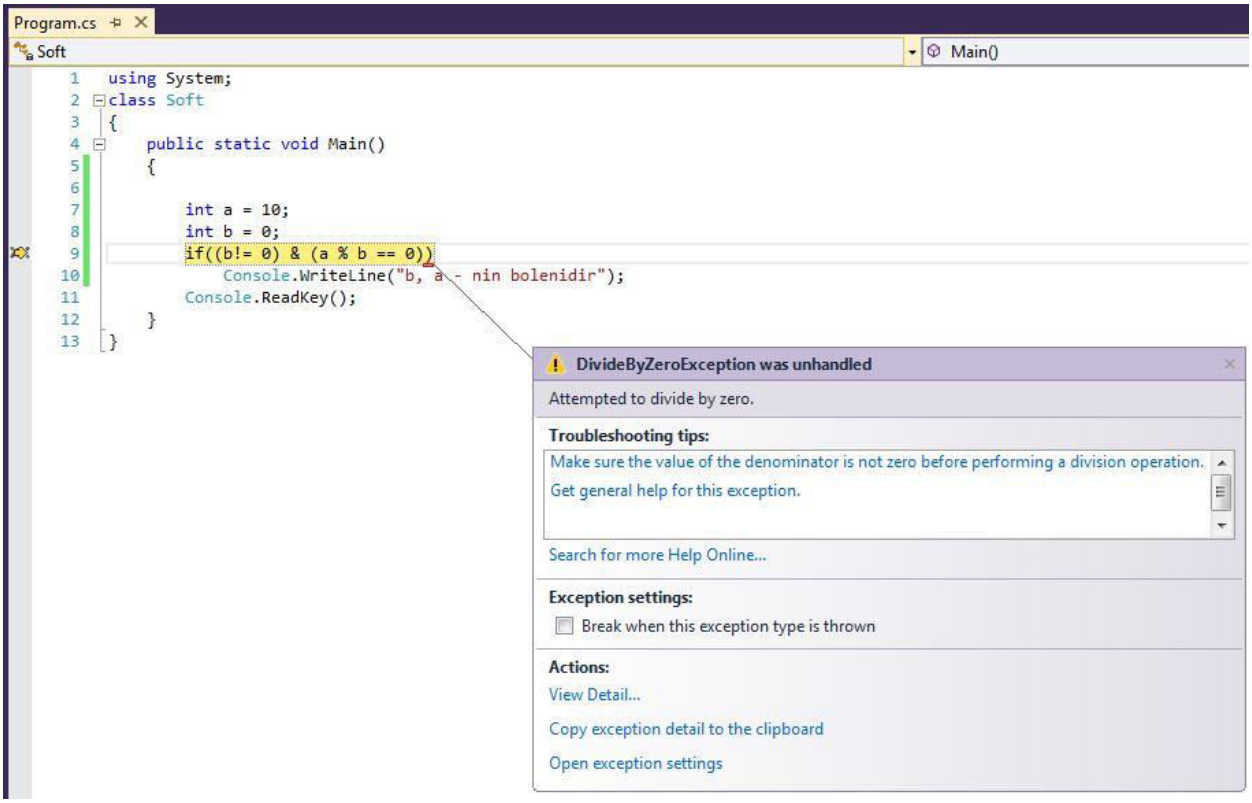
Şərtə bağlı məntiqi operatorlar && və || operatotlarıdır. Bu operatorlar da uyğun olaraq & və | operatoları kimi işləyir. Lakin onlardan kiçik bir fərqləri var. Məsələn, konkret olaraq & operatoruna baxaq. Əvvəlki cədvəldən də göründüyü kimi & operatorunun nəticəsinin false olması üçün operandların heç olmasa hansısa birinin false olması kifayətdir. Yəni, soldakı operand false olarsa, sağdakının nə olmasından asılı olmayaraq cavab false olacaq. Amma & operatoru soldakının cavabı nə olarsa olsun sağdakını da nəzərə alır. Yəni soldakının false olmağı, ümumi nəticənin false olmağı deməkdir, amma & operatoru haqqsızlıq olmasın deyə sağdakə operandı nəzərə alır. Amma belə baxanda soldakı operand false olarsa, sağdakını nəzərə almağa nə lüzum var? Onsuz da cavab false olacaq. Niyə əlavə vaxt itirib sağdakı operandın cavabının hesablanması üçün əməliyyat yerinə yetirək? Məhz bu işi yerinə yetirmək üçün & operatoruna alternativ olan && operatoru mövcuddur. Beləliklə && operatoru əgər soldakı operand false olarsa sağ tərəfə ümumiyyətlə "baxmır". Əgər sol operand true olarsa, məlumdur ki, sağdakı hökmən nəzərə alınacaq. Bu fərqi aşağıdakı proqram daha gözəl ifadə edir. Bu proqramda b dəyişəninin a – nın böləni olub-olmadığı yoxlanılır.

```
using System;
class Soft
{
   public static void Main()
   {

      int a = 10;
      int b = 0;
      if((b!= 0) & (a % b == 0))
         Console.WriteLine("b, a - nin bolenidir");
      Console.ReadKey();
```

```
    }
}
```

if((b!= 0) & (a % b == 0)) sətrinə fikir verək. Əvvəlcə b dəyişəninin sıfırdan fərqli olmağı yoxlanılır (b!= 0). Çünki sıfıra bölmə yoxdur. Sonra isə a dəyişənin b - ə qalıqsız bölünüb-bölünmədiyi yoxlanılır (a % b == 0). Bu münasibətdə b!= 0 şərti səhvdir, beləliklə if kontrol ifadəsinin şərti ödənmir və deməli b, a – nın böləni deyil. Lakin, & operatoru öz ənənəsinə sadiq qalaraq sağ tərəfi də nəzərə alır və a % b == 0 əməliyyatı yerinə yetirilməyə çalışır. b sıfır olduğu üçün isə, bu hissədə sıfıra bölmə xətası ilə qarşılaşırıq:




Lakin, & operatorunun yerinə && yazsaydıq, b! = 0 şərti səhv olduğu üçün sağ tərəf nəzərə alınmayacaqdı, deməli a % b == 0 əməliyyatı hesablanmayacaq və proqram bizə xəta verməyəcəkdi. Proqramın tam halı aşağıdakı kimi olacaqdır:

```
using System;
class Soft
{
   public static void Main()
   {

      int a = 10;
```

```
        int b = 0;
        if((b!= 0) && (a % b == 0))
            Console.WriteLine("b, a - nin bolenidir");
        else Console.WriteLine("b, a - nin boleni deyil");
        Console.ReadKey();
    }
}
```

|| operatoru üçün də tamamilə eyni sözləri demək olar. Yəni ||operatoruna görə, əgər sol tərəf true olarsa, sağ tərəfin nə olmasından asılı olmayaraq ümumi nəticə true olacaq. Göründüyü kimi bu operatorlar bəzi hallarda ancaq bir tərəfi – bir operandı nəzərə alır. Buna görə onlara şərtə bağlı operatorlar deyilir. Onu da qeyd edək ki, köhnə nəzəriyyələrdə bu operatorlara "qısa qapanma məntiqi operatorlar" da deyilir.

## Mənimsətmə operatoları

Mənimsətmə əməliyyatı proqramlaşdırmada ən fundamental və vacib əməliyyatlardan biridir. Mənimsətmə deyəndə bir dəyişənə (=) operatorunun vasitəsilə bir qiymətin verilməsi başa düşülür. Mənimsətmənin ümumi sintaksisi belədir:

Dəyişən = qiymət;

Aşağıdakı ifadə tamamilə düzgündür:

int a, b, c;

a = b = c = 44;

Bu kod sətirləri a, b, c dəyişənlərinin hər birinə 44 qiymətini verir.

### Bitişik mənimsətmələr

Aşağıdakı mənimsətmə ifadəsinə baxaq:

int a = 5;

a = a + 10;

Burada a dəyişəninə onun qiymətinin 10 ilə toplanmasından alınan qiyməti mənimsətdik. Bu mənimsətməni aşağıdakı kimi daha kompakt şəkildə yaza bilərik:

a += 10;

Bura da += operatoruna fikir verin, bu operator a dəyişəninə onun qiymətinin 10 vahid artığını mənimsədir. Bu cür mənimsətdə yuxarıdakına nisbətən daha professional

yazılışdır və daha sürətidir. Bununla yanaşı aşağıdakı belə mənimsətmə operatorları da var:

+=

-=

*=

\=

&=

|=

və s. Məsələn

a = a * 5 ifadəsi

a *= 5

ifadəsinə bərabərdir. Bu operatorlara bitişik mənimsətmə operatorları deyilir.

## Bit Əsaslı Operatorlar

C# - da mövcud digər operatorlar qrupundan biri də bit əsaslı operatorlardır.Bu operatorlar birbaşa operandarın bitləri üzərində işləməyimizə imkan verir və bu operatorların operandları tam olmalıdır. Yəni float, bool, double kimi verilənlər tipinə aid dəyişənlər üzərində bit operatorlarla işləyə bilmərik.

Bu operatorlara ona görə bit əsaslı operatorlar deyilir ki, bu operatorlar bir ədədin bitləri, yəni ikkilik sistemdəki '0' - ları və '1' – ləri üzərində işləyir. Bit əsaslı operatorlar aşağıdakılardır:

| Operator | Mənası |
|---|---|
| & | Bit əsaslı VƏ |
| \| | Bit əsaslı VƏYA |
| ^ | Bit əsaslı XOR |
| ~ | Bit əsaslı inkar |

Bu operatorların tətbiqinin nəticələrini nəzərə alaraq aşağıdakı cədvələ baxaq:

| a | b | a&b | a\|b | a^b | ~a |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Məsələn 10 & 13 əməliyyatına baxaq. Qeyd olunduğu kimi, operatorlar bitlərlə işlədiyi üçün prosesi aydın görə bilmək açısından bu ədədləri əvvəlcə ikilik sistemə çevirək:

10 = 1 0 1 0

13 = 1 1 0 1

Beləliklə,

```
   1 0 1 0
&  1 1 0 1
-----------
   1 0 0 0 = 8
```

Deməli, 13 & 10 = 8 edirmiş.

İndi bir proqram nümunəsinə baxaq. Əlinizə qələm, vərəq alıb bir neçə tək və cüt ədədi ikilik sistemə çevirsəniz görərsiniz ki, bütün tək ədədlərin ikilik sistemdəki sonuncu biti 1, bütün cüt ədədlərin sonuncu biti isə 0- dır. Bu xüsusiyyətdən istifadə edərək, tək ədədi cüt ədədə çevirən bir proqram yaza bilərik. Etməli olduğumuz şey, sadəcə ədədin sonuncu bitini 0 – a çevirməkdən ibarət olacaq. Ədədin sonuncu bitindən əvvəlki bitlərinə toxunmamaq üçün, ...11111110 şəklində bir ədəd seçək və bu ədədlə müvafiq ədədə bit əsaslı VƏ operatorunu tətbiq edək.

```
using System;

class Soft
{
   public static void Main()
   {
      for (int i = 0; i <= 10; i++)
      {
```

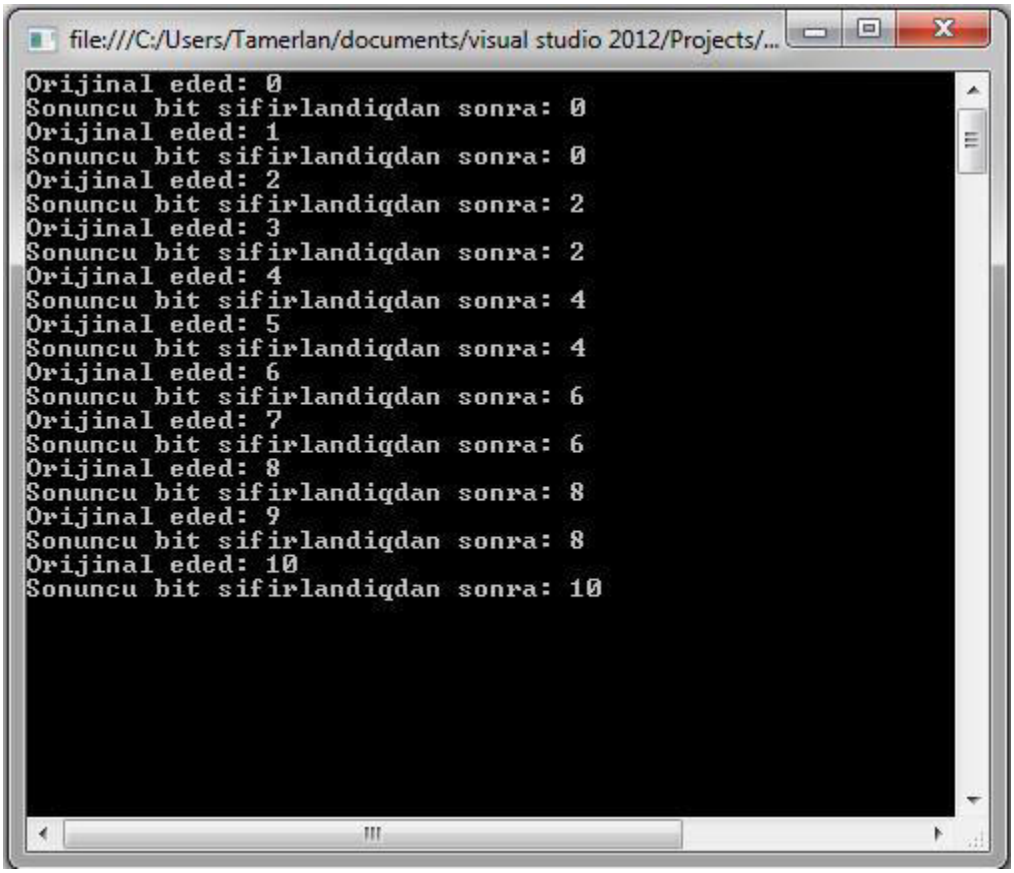
```
            Console.WriteLine("Orijinal eded: " + i);
            Console.WriteLine("Sonuncu bit sifirlandiqdan sonra: " + (i & 65534));
        }
        Console.ReadKey();
    }
}
```

Burada 65534  ədədinin ikilik sistemdə təsviri

11111110

kimidir.

Bu proqramda 0 – dan 10 – a kimi dövr qurulur və hər i ədədi 65534 ədədi ilə bit əsaslə VƏ operatorundan keçirilir. Beləliklə ədəd cütdürsə, bu ədədin sonuncu biti onsuz da 0 – dır. Yəni, bir dəyişiklik olmayacaq, ədəd tək olduqda isə sonuncu bit 1 olduğundan 1 & 0 = 0 edir və ədəd bir vahid azalır – cüt ədədə çevrilir. Beləliklə proqramın nəticəsi aşağıdakı kimidir:



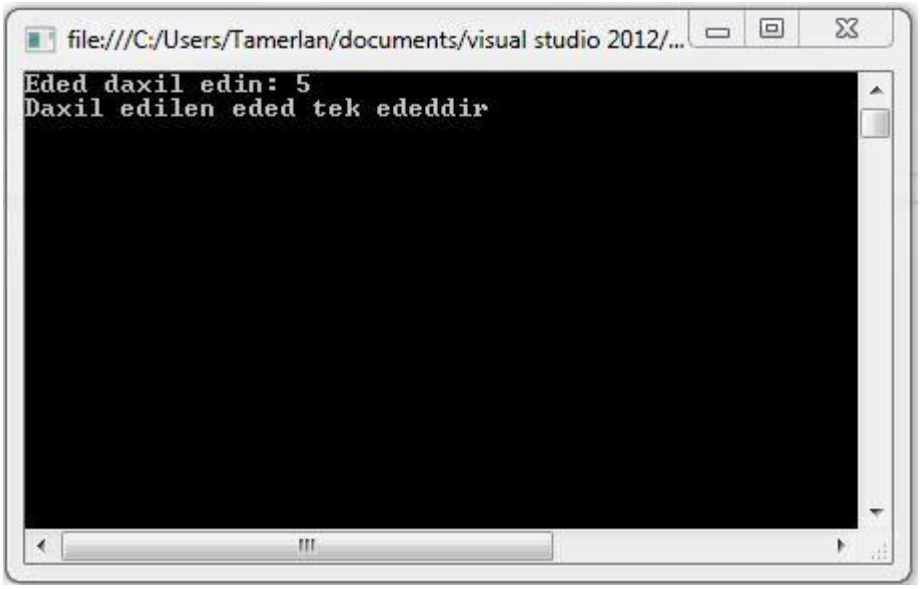

Bu üsuldan istifadə edərək daxil edilən ədədin tək və ya cüt olduğunu müəyyənləşdirə bilərik. Artıq bilirik ki, cüt ədədlərin ikilik sistemdəki sonuncu biti 0 – dır. Beləliklə, daxil

edilən ədədlə …000001 şəklində bir ədədə bit əsaslı VƏ operatorunu tətbiq edək. Əgər nəticə 0 olarasa deməli ədəd cütdür (yəni sonuncu bit sıfırdır). Əgər nəticə 1 olsa deməli ədəd təkdir:

```
using System;

class Soft
{
    public static void Main()
    {
        Console.Write("Eded daxil edin: ");
        int eded = Convert.ToInt32(Console.ReadLine());
        if((eded & 1) == 0)
            Console.WriteLine("Daxil edilen eded cut ededdir");
        else
            Console.WriteLine("Daxil edilen eded tek ededdir");
        Console.ReadKey();
    }
}
```

Proqram aşağıdakı nəticəni verəcək:




Bit əsaslı VƏYA operatoru da oxşar qaydada işləyir. Bit əsaslı XOR operatoruna baxaq. Bu operatorun çox maraqlı bir tərəfi vardır. Cədvəldən də göründüyü kimi, bu operatorun operandları bir-birlərindən fərqli olarsa, nəticə 1 olur, əks halda 0 olur. Belə ki, əgər a ədədinə b ədədini XOR edib c ədədini alırıqsa, c ədədinə yenidən b ədədini XOR etsək, a ədədini – yəni orijinal ədədin özünü alacağıq. Ağlınıza nə gəldi? Bu üsuldan istifadə edərək, bir şifrələmə proqramı yaza bilərik mi? Əgər b ədədini bir şifrə

kimi götürsək, a məlumatını b ilə şifrələyə, sonra yenə b ilə deşifrələyə bilərik. Aşağıdakı proqrama baxaq:

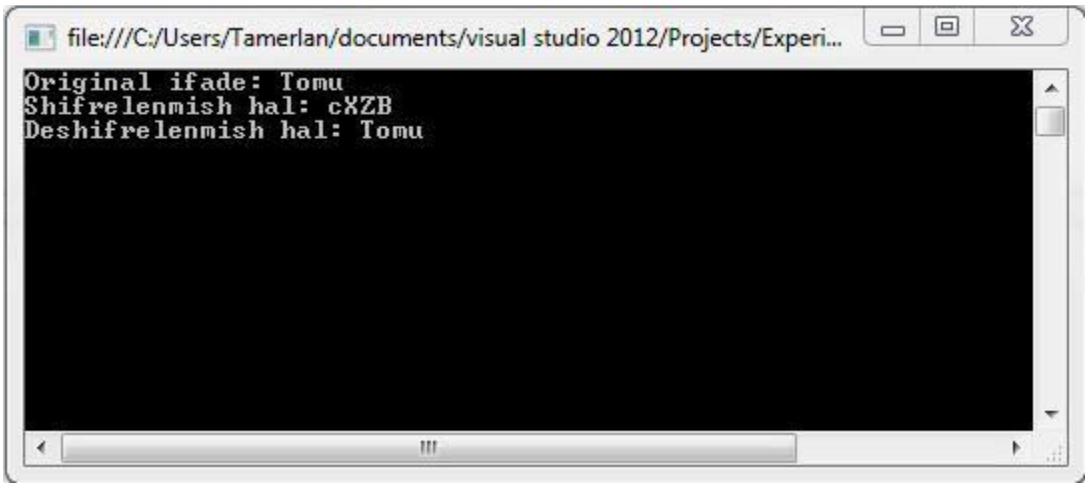
```
using System;

class Soft
{
   public static void Main()
   {
      int parol = 55;
      char c1 = 'T';
      char c2 = 'o';
      char c3 = 'm';
      char c4 = 'u';
      Console.WriteLine("Original ifade: " + c1 + c2 + c3 + c4);

      //Shifreleyek
      c1 = (char)(c1 ^ parol);
      c2 = (char)(c2 ^ parol);
      c3 = (char)(c3 ^ parol);
      c4 = (char)(c4 ^ parol);
      Console.WriteLine("Shifrelenmish hal: " + c1 + c2 + c3 + c4);

      //Deshifreleyek
      c1 = (char)(c1 ^ parol);
      c2 = (char)(c2 ^ parol);
      c3 = (char)(c3 ^ parol);
      c4 = (char)(c4 ^ parol);
      Console.WriteLine("Deshifrelenmish hal: " + c1 + c2 + c3 + c4);
      Console.ReadKey();
   }
}
```

Deməli hər bir simvolun Unicode massivindəki sıra nömrəsi ilə (sıra nömrəsi tam ədəddir) 55 ədədini (parol dəyişənin qiyməti) XOR əməliyyatından keçirdik və başqa bir ədəd aldıq. Sonra Unicode massivində alınmış ədədə uyğum simvolu ekranda əks etdirdik. Beləliklə T simvolunun nömrəsi ilə 55 – i XOR edəndə char qarşılığı "c" olan simvol əldə olunur. Sonra bu simvolun sıra nömrəsi ilə 55 ədədini yenidın XOR edəndə bu zaman da əvvəlki – orijinal simvolun sıra nömrəsinə uyğun ədəd alırıq və həmin ədədə uyğum simvolu əks etdiririk. Bu əməliyyatı "Tomu" sətrinin bütün simvolları üçün tətbiq edirik. Beləliklə, proqramın nəticəsi aşağıdakı kimi olur:




## ? Operatoru

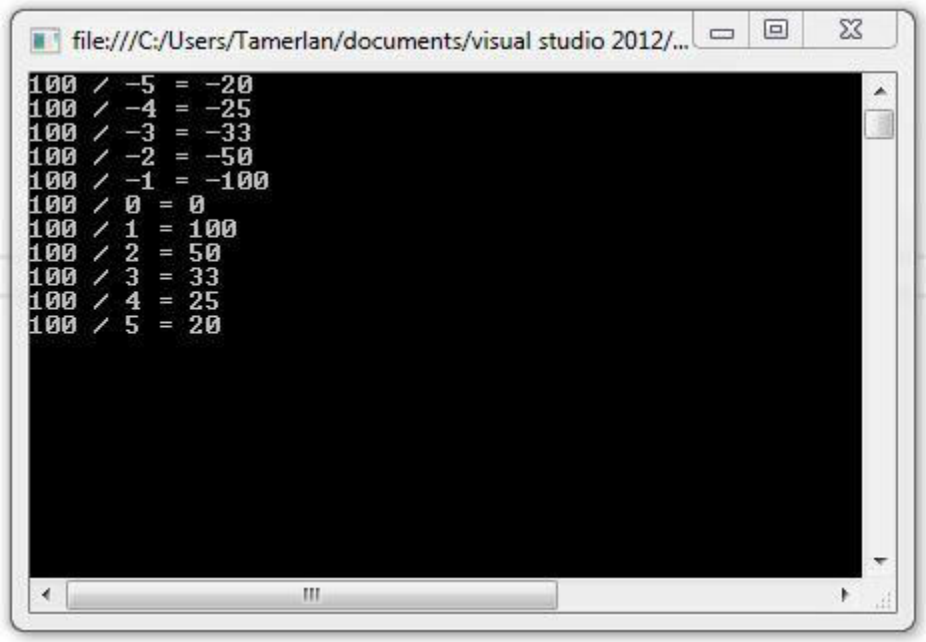
C# - ın ən maraqlı operatorlarından bri də ? operatorudur. Bu operatora üçlü operator da deyilir. Çünki, operator üç ədəd operand tələb edir. Bu operator "əgər – onda- əks halda" kombinasiyasına alternativ yaradılıb. ? operatorunun ümumi sintaktik şəkli aşağıdakı kimidir:

Ifadə1 ? ifadə2 : ifadə3;

Burada ifadı1 bool tipində bir ifadədir. Əgər bu ifadə true qiymətinə malikdirsə, onda ? operatorunun nəticəsi ifadə2 olur, əks halda nəticə ifadə3 olur. Bir proqram nümunəsinə baxaq. Bu proqramda -5 -dən 5 - ə qədər dövr qurulacaq və hər bir aradakı ədədə, 100 ədədinin bölünməsindən alınan nəticə ekranda əks olunacaq. [-5 ; 5] parçasında 0 ədədi də yerləşdiyi üçün sıfıra bölmənin qarşısını ? operatorunun ifadə1 şərti ilə alaq. Beləliklə proqram aşağıdakı kimi olacaq:

```
using System;
class Soft
{
   public static void Main()
   {
      int netice;
      for (int i = -5; i <= 5; i++)
      {
         netice = i != 0 ? 100 / i : 0;
         Console.WriteLine("100 / " + i + " = " + netice);
      }
      Console.ReadKey();
   }
}
```

Bu proqramın nəticəsi aşağıdakı kimidir:




Bu proqramda

netice = i != 0 ? 100 / i : 0;

sətrinə fikir verin. Bu sətrin mənası belədir: "**Əgər** i sıfırdan fərqlidirsə, **onda** netice dəyişəninə 100 / i ifadəsini mənimsət, **əks halda** netice dəyişəninə 0 mənimsət". Yəni buradakı 0, bizim təyin etdiyimiz ixtiyari qiymətdir, sadəcə i sıfır olduqda 100 / i ifadəsinin hesablanmamağı üçün (çünki hesablansa xəta verəcək), 100 / 0 ifadəsinə "boş" qalmasın deyə 0 mənimsətdik.

## Operatorların Öncəlik Sırası

Mən 4- cü sinifdə oxuyurdum (2004 –cü il), ilik ortalarına yaxın sinif rəhbərimiz bizə riyaziyyat dərsində mötərizələr daxil olan ifadələri keçdi. Onun bu sözləri hələ də yadımdadır: "Uşaqlar, birinci vurma, bölmə hesablanır, sonra toplama və çıxma. Əgər misalda mötərizə varsa, əvvəlcə mötərizənin içi hesablanır". Bu qayda riyaziyyatın ən fundamental qaydası olduğu kimi, proqramlaşdırmada da belədir. Yəni, müxtəlif operatorların daxil olduğu bir ifadədə əvvəlcə mötərizənin içi hesablanır, sonra *, / operatorlarına aid hissələr hesablanır. Məsələn,

int eded = 5 + 2 * 10;

ifadəsinin nəticəsi ilə,

int eded = (5 + 2) * 10;

ifadəsinin nəticəsi çox fərqlidir. Birinci ifadə 2 ilə 10 ədədinin hasilini hesablayır və üzərinə 5 əlavə edir, beləliklə cavab 25 olur. İkinci ifadədə mötərizə üstünlük təşkil etdiyi üçün, əvvəlcə 5 ilə 2 cəmlənir, alınan nəticə 10 ədədinə vurulur, beləliklə, nəticə 70 olur.

C# - da operatorların öncəlik sırası, yüksəkdən alçağa doğru aşağıdakı kimi yazıla bilər.

Ən yüksək:

() []

! ~

* /

+ -

< >

== !=

&

^

|

&&

||

?:

=

Ən aşağı

# BÖLMƏ 4. PROQRAM KONTROL İFADƏLƏRİ

Proqram kontrol ifadələri, proqramın icrası zamanı axışı idarı edən, onları orqanizasiya edən ifadələrdir. C# - da proqram kontrol ifadələri (Program Control Statements) 3 kateqoriyaya bölünür:

- Şərf ifadələri: if, switch

- Dövr ifadələri: for, while, do-while, foreach

- Dəyişdirmə ifadələri: break, continue, goto, return, throw

## if Şərt İfadəsi

Biz proqramımızı işə salanda, proqramımız yuxarıdan aşağı doğru (Main metodundan başlayaraq) sətir-sətir icra olunmağa başlayır. Bəzən vəziyyət elə olur ki, proqramın müəyyən hissəsinin icra olunub – olmaması, hansısa şərtə bağlı olsun. Şəni müəyyən bir şərt daxlində poqramın bir hissəsi icra olunsun, ya da icra olunmasın. Bu əməliyyatı həyata keçirmək üçün if şərt ifadısindən istifadə edəcəyik. if ifadəsinin vəzifəsi ondan ibarətdir ki, hansısa bir şərtin doğru olduğu təqdirdə, hansısa kodlar icra olunsun, əks halda – şərt düzgün olmadığı halda həmin kodlar icra olunmasın Bu ifadənin bötüv sintaktik şəkli aşağıdakı kimidir:

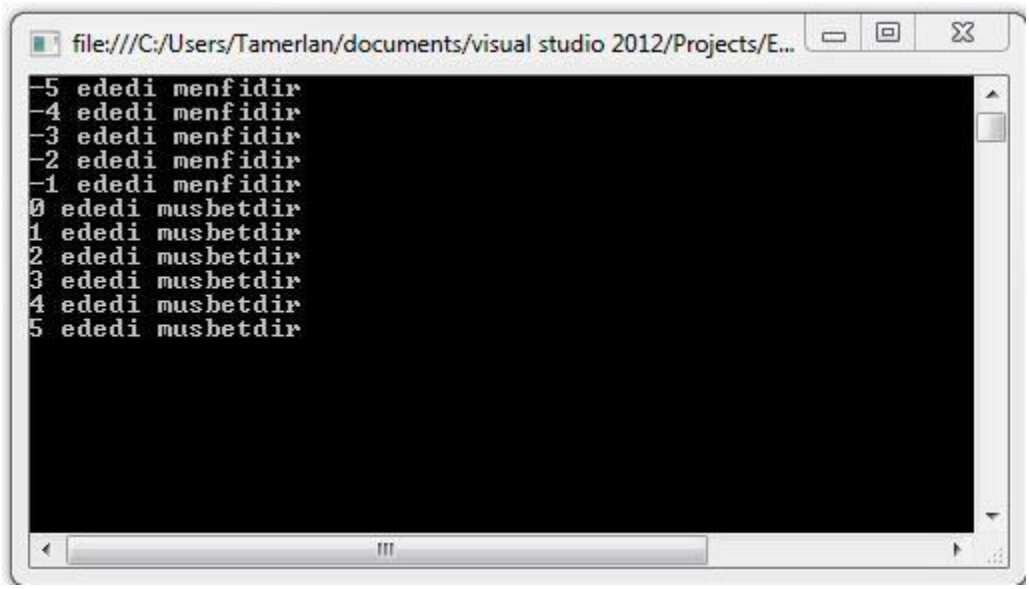
```
if(şərt)
{
    Əməliyyatlar
}
else
{
     Əməliyyatlar
}
```

Burada şərt doğru olsa (true) onda birinci fiqurlu mötərizə blokunun içərisi icra olunur. Əks halda ikinci blok, yəni else hissəsinə aid olan blok icra olunur. Heç bir zaman hər iki blok eyni anda icra oluna bilməz. Burada **şərt** bool tipində bir qiymətdir. Yalnış olsa, yəni false, gövdə icra olunmayacaq. Qeyd edək ki, yerinə yetiriləcək əməliyyatlar, sadəcə bir sətifdən ibarətdirsə, onda fiqurlu mötərizələr yazılmaya da bilər. Aşağıdakı misala baxaq:

```
using System;
class Soft
```

```
{
    public static void Main()
    {
        int i;
        for (i = -5; i < 6; i++)
        {
            if (i < 0) Console.WriteLine( i + " ededi menfidir");
            else Console.WriteLine(i + " ededi musbetdir");
        }
        Console.ReadKey();
    }
}
```

Nəticə, məlumdur ki, aşağıdakı kimi olacaq:




Burada i dəyişəni mənfi qiymət alarsa if - ə aid blok icra olunacaq, əks halda else hissəsinə aid blok icra olunacaq.

## İç-içə if ifadələri

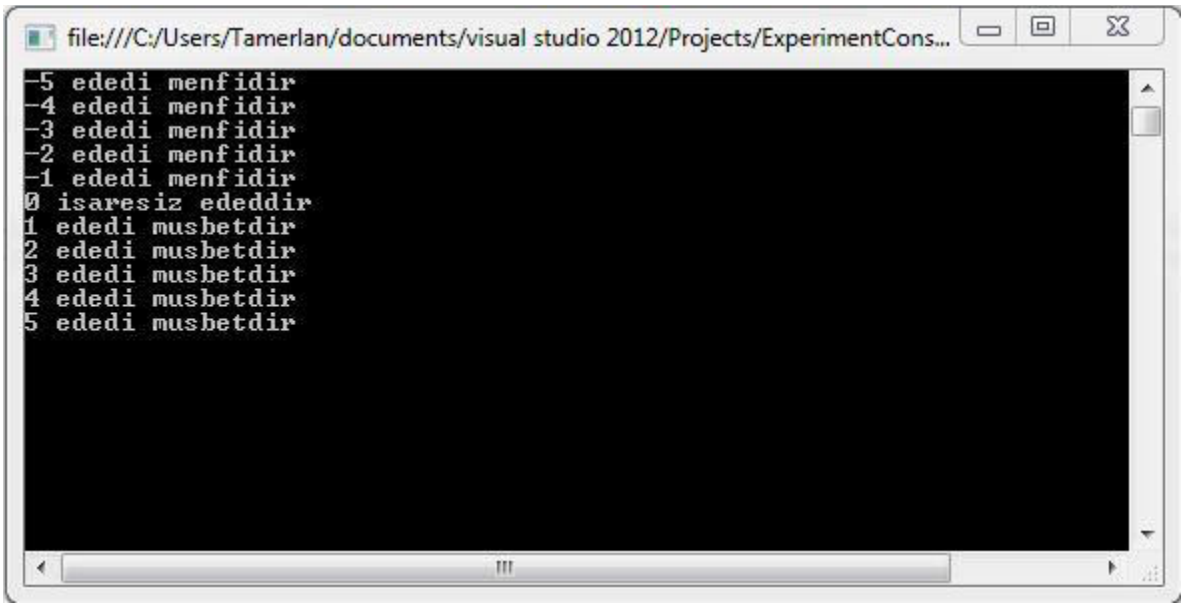
İç-içə if (nested if) ifadələri, dedikdə bir if və ya else ifadəsinin blokunun içərisində başqa bir if ifadəsinin olması başa düşülür. Bu tip yazılışlar proqramlaşdırmada çox istifadə olunur. Amma bu tip iç-iç if ifadələrindən istifadə edərkən, unutmamaq lazımdır ki, hər if özünə uyğun else ilə başlanır. Yəni, çöldəki if- ə aid else hissəsinin, içəridəki if- ə aid else hissəsinə heç bir dəxli yoxdur. Məsələn aşağğıdakı kod parçasına baxaq:

```
if (10 > 9)
{
    if(5 < 3)
        Console.WriteLine("5 < 3 = true");
    else Console.WriteLine("5 < 3 = false");
}
else Console.WriteLine("10 < 9 = false");
```

Burada içəridəki if ifadəsinin 5 < 3 şərti doğru olmadığı üçün, ona aid içəridəki else hissəsinə aid sətir icra olunacaq və ekrana "5 < 3 = false" ifadəsi yazılır. Yəni, içəridəki if ifadəsinin şərtinin düzgün olmamağı, heç bir zaman ekrana "10 < 9 = false" yazılmasına səbəb ola bilməz. Yuxarıdakı proqramda -5 dən 5- ə qədə mənfi və müsbət ədədləri bir-birlərindən ayırdıq. Burada 0 müsbət ədəd kimi qeyd olundu. Amma riyaziyyatda 0 nə müsbət, nə də mənfi ədəd kimi qeyd olunur. İç-içə if ifadəsindən istifadə edərək, yuxarıdakı proqramı belə yaza bilərik:

```
using System;
class Soft
{
    public static void Main()
    {
        int i;
        for (i = -5; i < 6; i++)
        {
            if (i < 0) Console.WriteLine(i + " ededi menfidir");
            else if (i == 0) Console.WriteLine("0 isaresiz ededdir");
            else Console.WriteLine(i + " ededi musbetdir");
        }
        Console.ReadKey();
    }
}
```

Nəticə aşağıdakı kimi olacaq:




## if-else-if kombinasiyası

İç-içə yerləşmiş if ifadəsinə söykənən və çox istifadə olunan bir şərt kontrol mexanizmi də **if-else-if** kombinasiyasıdır. Bu yazılışın ümumi sintaktik şəkli aşağıdakı kimidir:

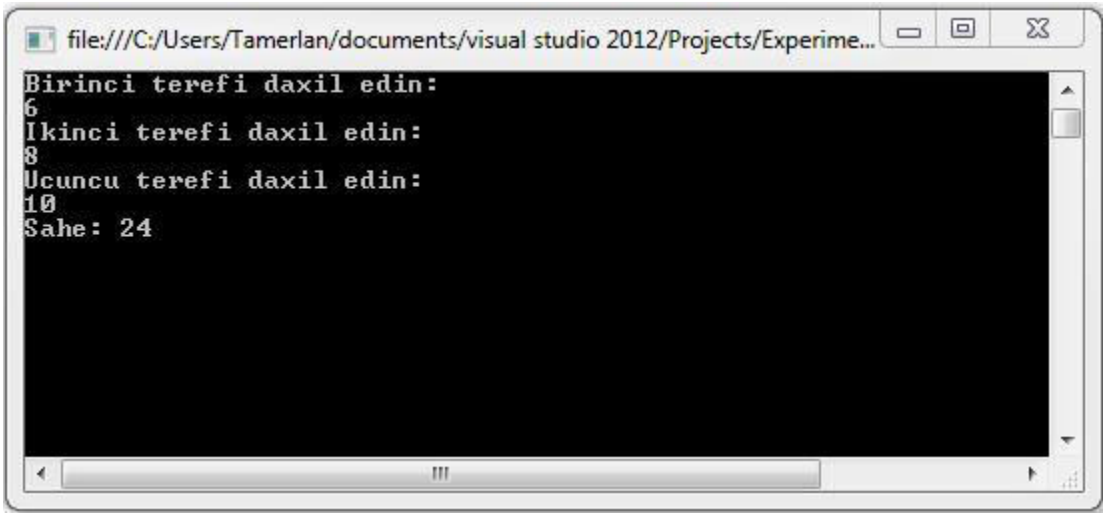
```
if(şərt)
    əməliyyat;
else if(şərt)
    əməliyyat;
else if(şərt)
    əməliyyat;
.
.
.
if(şərt)
    əməliyyat;
else…
```

Bu kombinasiyada əməliyyat yuxarıdan-aşağıya doğru yerinə yetirilir. Belə ifadələrdə, bir if ifadəsinin yerinə yetirilməsi üçün, ondan yuxarıda yerləşən if ifadəsinin şərtinin yanlış olması lazımdır. Bu kombinasiyanı sözlə ifadə etsək: "Əgər bir şərt düzgündürsə,

əməliyyat yerinə yetir, əks halda başqa bir şərti yoxla, əgər o düzgündürsə, onda əməliyyat yerinə yetir, əks halda yenə yenə başqa şərti yoxla…" .İndi bir proqrama baxaq. Bu proqramda daxil edilən 3 ədədə görə düzbucaqlı üçbucağın sahəsini hesablayan proqram yazacağıq. Düzbucaqlı üçbucaq hər hansı iki tərəfi 90 dərəcə bucaq altında kəsişən üçbucaqlardır. Bu üçbucağın sahəsi, onun düz bucaq altında kəsişən tərəflərinin hasilinin yarısına bərabərdir. Bu tərəflərə kated, digər tərəfə isə hipotenus deyilir. Pifaqor teoreminə görə düzbucaqlı üçbucağın katedlərinin kvadratları cəmi, onun hipotenusunun kvadratına bərabər olur. Bu teoremdən istifadə edərək, daxil edilən üç tərəfin hansıların kated olduğunu tapacağıq. Əgər bu ədədlər üzərində teoremin şərtləri ödənilməzsə, deməli daxil edilən üç ədəd, hansısa düzbucaqlı üçbucağın tərəflərinə uyğun gəlmir. Bununla yanaşı, üçbucaq bərabərsizliyinə görə istənilən üçbucağın ixtiyari iki tərəfinin cəmi digər tərəfdən böyük olmalıdır. Yəni, daxil edilən ədədlərin, düzbucaqlı üçbucaqdan ziyadə ümumiyyətlə bir üçbucağın tərəflərinə uyğun gəlib- gəlmədiyini yoxlayacağıq. Digər tərəfdən, üçbucağın tərəfləri məsafə anlayışını ifadə etdiyi üçün, mənfi ola bilməzlər. Bunların hamısını nəzərə alaraq, professionla yaxın bir sahə hesablayan proqram yazaq:

```
using System;
class Soft
{
    public static void Main()
    {
        Console.WriteLine("Birinci terefi daxil edin: ");
        int teref1 = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Ikinci terefi daxil edin: ");
        int teref2 = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Ucuncu terefi daxil edin: ");
        int teref3 = Convert.ToInt32(Console.ReadLine());
        if (teref1 < 0 | teref2 < 0 | teref3 < 0) //(1)
            Console.WriteLine("Terefler menfi ola bilmez");
        else
        {
            if ((teref1 + teref2 < teref3) | (teref1 + teref3 < teref2) | (teref2 +
teref3 < teref1)) //(2)
                Console.WriteLine("Daxil etdiyiniz ededler, hansisa ucbucagin tereflerine
uygun gelmir");
            else //(3)
            {
                if (teref1 * teref1 + teref2 * teref2 == teref3 * teref3)
                    Console.WriteLine("Sahe: " + (teref1 * teref2) / 2);
                else if (teref1 * teref1 + teref3 * teref3 == teref2 * teref2)
                    Console.WriteLine("Sahe: " + (teref1 * teref3) / 2);
                else if (teref2 * teref2 + teref3 * teref3 == teref1 * teref1)
                    Console.WriteLine("Sahe: " + (teref2 * teref3) / 2);
                else Console.WriteLine("Terefler duzbucaqli ucbucaga uygun gelmir");
            }
        }
        Console.ReadKey();
    }
}
```

(1) sətrində daxil edilən tərəflərin işarəsini yoxladıq. Əgər hamısı müsbətdirsə, onda (2) şərti ilə üçbucaq bərabərsizliyini yoxladıq. Əgər daxil edilən tərəflər hansısa üçbucağa uyğundursa, (3) sətrinə aid blok ilə bu tərəflərin ümumiyyətlə hansısa düzbucaqlı üçbucağın tərəflərinə uyğun gəlib-gəlmədiyini yoxladıq. Nəticə aşağıdakı kimi olacaq:




## switch İfadəsi

switch kontrol ifadəsi də bir şərt ifadəsidir. Bu ifadə bir dəyişəni qiymətini, ardıcıl yerləşən sabitlərlə müqayisə edir və uyğunluq hallarında müəyyən birr əməliyyatı yerinə yetirməyə imkan verir. Əslində switch ifadəsinin yerinə yetirdiyi bütün işləri for və if kontrol ifadələrinin kombinasiyaları ilə də yerinə yetirmək olduğu halda, switch kontrol ifadəsi bir çox hallarda işimizi asanlaşdırır. Switch ifadəsinin ümumi sintaktik şəkli aşağıdakı kimidir:

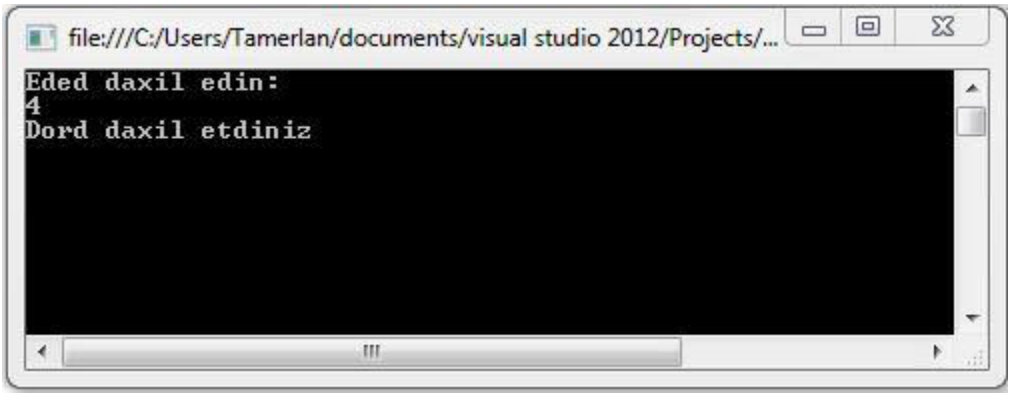
```
switch(ifadə)   {
    case sabit1: əməliyyatlar
        break;
    case sabit2: əməliyyatlar
        break;
     case sabit3: əməliyyatlar
        break;


    .
    .
    .
    case sabit2: əməliyyatlar;
        break;
    default: əməliyyatlar
        break;
}
```

Burada **ifadə** hesablanır və onun qiyməti blok içərisindəki sabitlərlə bir-bir müqayisə olunur. ifadənin nəticəsi int, char, bool, sbyte, byte, ulong kimi tiplərində olmalıdır. Kərs tipli nəticə ola bilməz. Əgər ifadənin qiyməti qeyd olunmuş **case** sabitlərinin birinə bərabər olarsa, onda həmin hissəyə aid **əməliyyatlar** icra olunur və switch ifadəsindən çıxılır. Proqramın axışı switch ifadəsinin son fiqurlu mötərizəsindən icra olunmağa davam edir. Əgər ifadənin qiyməti qeyd olunmuş sabitlərin heç birinə bərabər olmazsa, onda default hissəsinə aid əməliyyatlar icra olunur. switch ifadəsində default hissəsinin olması zəruri deyil, yəni default yazılmaya da bilər. Bu zaman heç bir uyğunluq olmazsa, onda switch ifadəsinə aid heç bir əməliyyat yerinə yetirilməyəcək. case qarşısındakı qiymətlər əsla bir dəyişənin qiyməti ola bilməz. Çünki switch ifadəsində case qarşısındakı qiymətlər bir-birlərindən fərqli olmalıdır. İki qiymətin eyni olmağı, uyğunluq zamanı iki case əməliyyatının yerinə yetirilməyideməkdir. Bu isə prinsipə ziddir. Dəyişənlərlərin qiymətləri də proqram daxilində dəyişə bildiklərindən, case qarşısındakı verilənlər mütləq sabitlər olmalıdır. Aşağıdakı proqramda switch ilə 1 ilə 5 arasında bir rəqəm daxil edilməsi istənilir və daxil edilən ədədin yazı variantı ekranda əks olunur:

```
using System;
class Soft
{
    public static void Main()
    {
        Console.WriteLine("Eded daxil edin: ");
        int eded = Convert.ToInt32(Console.ReadLine());
        switch (eded)
        {
            case 1: Console.WriteLine("Bir daxi etdiniz");
                break;
            case 2: Console.WriteLine("Iki daxi etdiniz");
                break;
            case 3: Console.WriteLine("Uc daxil etdiniz");
                break;
            case 4: Console.WriteLine("Dord daxil etdiniz");
                break;
            case 5: Console.WriteLine("Besh daxil etdiniz");
                break;
            default: Console.WriteLine("Daxil etdiyiniz eded [1; 5] parchasinda deyil");
                break;
        }
        Console.ReadKey();
    }
}
```

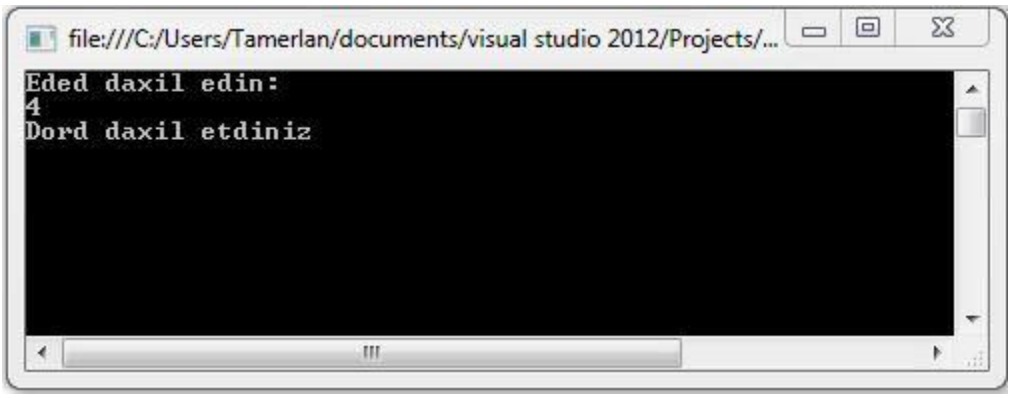
Daxil edilən ədəd 1, 2, 3, 4, 5 sabitlərinin hər biri ilə müqayisə olunur və bərabərlik zamanı müvafiq əməliyyat yerinə yetirilir:




Qeyd etmək lazımdır ki, C# - da switch ifadəsində hər case əməliyyatı break ilə sonlanmalıdır. Buna "növbəti addıma sürüşməmə" (no fall through) qaydası deyilir. Yəni, hət case əməliyyatının bir brake – i olmalıdır. Bununla belə switch ifadəsinin qiymətini

bir neçə sabitlə də müqayisə etmək mümkündür. Aşağıdakı kod parçası tamamilə düzgündür:

```
using System;
class Soft
{
   public static void Main()
   {
      Console.WriteLine("[1; 10] arasinda eded daxil edin: ");
      int eded = Convert.ToInt32(Console.ReadLine());
      switch (eded)
      {
         case 2:
         case 4:
         case 6:
         case 8:
            Console.WriteLine("Cut eded daxil etdiniz");
            break;
         default: Console.WriteLine("Te eded daxil etdiniz");
            break;
      }
      Console.ReadKey();
   }
}
```




## for dövr ifadəsi

Proqramın axışını idarə edən növbəti kontrol ifadələrindən biri də dövr ifadələridir. Bəzən vəziyyət elə olur ki, poqramda bir əməliyyatı müəyyən sayda yerinə yetirmək – təkrar icra etmək lazım gəlir. Bu kimi bir işi dövrə salaraq təkrarən yerinə yetirmək üçün dövr kontrol ifadələrindən istifadə olunur. Bu ifadələrdən biri də **for** – dur. for ifadəsinin ən çox istifadə olunan sintaktik şəkli aşağıdakı kimidir:

```
for(başlanğıc; şərt; ifadə)

{

    Əməliyyatlar…

}
```

Burada **başlanğıc** adətən, dövrü idarə edən dəyişənə müəyyən bir başlanğıc qiymətin verilməsi kimi müəyyən olunur. **Şərt**, dövrün hansı şərtlər daxilində icra olunmasını müəyyənləşdirən şərtdir və ümumi qiyməti bool tipində olur. Nə qədər ki, bu şərt doğrudur, dövr icra olunur. **ifadə** isə for ifadəsi hərdəfə dövr etdikcə, dövr idarə edən dəyişənin hansı şəkildə dəyişəcəyini özündə saxlayır. Qeyd etmək lazımdır ki, dövrün şərti ilk addımda düzgün deyilsə, onda dövr ifadəsinə aid heç bir əməliyyat yerinə yetirilmir. Aşaşağıdakı nümunəyə baxaq:

```
using System;
class soft
{
    static void Main()
    {
        for(int i = 0; i > 5; i++)
        {
            Console.WriteLine(i);
        }
        Console.ReadKey();
    }
}
```

Burada dövrü idarəedən dəyişən i dəyişənidir. Göründüyü kimi, i dəyişənin bağlanğıc qiyməti sıfırdır və bu dəyər 5 – dən böyük deyil. Yəni, bu proqramda ekrana heç bir şey çıxmayacaq.

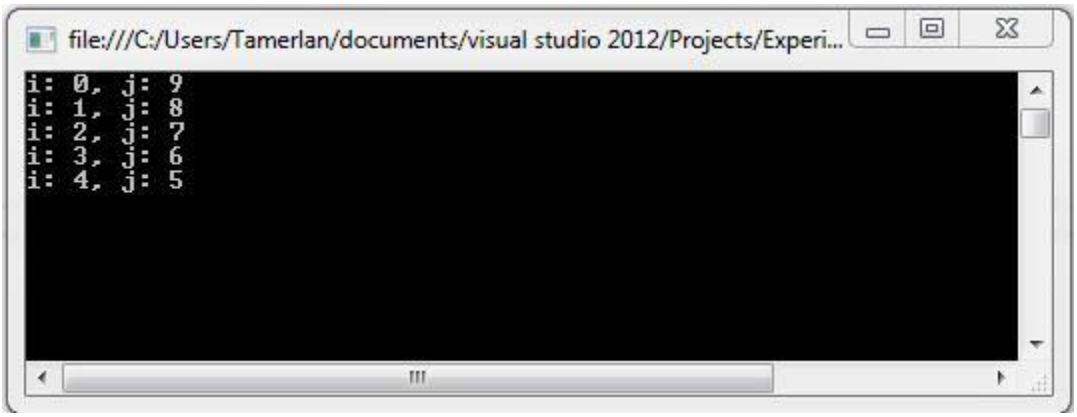
## Birdən çox dövr idarəedən dəyişən

Əvvəlcə aşağıdakı proqram nümunəsinə baxaq. Bu proqramda iki dəyişənin qiymətləri bir-birləri ilə müqayisə olunaraq, dəyişdiriləcək:

```
using System;
class soft
{
    static void Main()
    {
        int j = 10;
        for(int i = 0; i < 10; i++)
```

```
        {
            if (i < j)
            {
                j--;
                Console.WriteLine("i: {0}, j: {1}", i, j);
            }
        }
        Console.ReadKey();
    }
}
```

Deməli, bu proqramda i dəyişəninin qiyməti j dəyişəninin qiyməti ilə müqayisə olunur, i artırılır, j isə azaldılır və i < j şərti daxilində müvafiq qiymətlər ekranda ks olunur. Odur ki, proqramın nəticəsi aşağıdakı kimi olur:




Bu əməliyyatı, dövr kontrol ifadəsi içərisində iki dəyişən yazaraq, aşağıdakı kimi çevirə bilərik:

```
using System;
class soft
{
    static void Main()
    {

        for(int i = 0, j = 10; i < j; i++, j--)
        {
                Console.WriteLine("i: {0}, j: {1}", i, j);
        }
        Console.ReadKey();
    }
}
```

Burada

for(int i = 0, j = 10; i < j; i++, j--)
sətrinə fikir verin. Bir dövr kontrol ifadəsi içində, birdən çox dövr idarəedən dəyişən yaza bilərik. Bu dövr i < j şərti doğru olduğu müddətdə, hər dövrdə i dəyişənini bir vahid artırır, j dəyişənini isə bir vahid azaldır və i = 5, j = 5 olduqda, artıq şərt pozulur və dövr saxlanılır. Bu proqramın da ekran nəticəsi tamamilə eynidir. Göründüyü kimi, əgər dövr idadəedən dəyişən birdən çoxdursa, onda onlar bir-birlərindən vergül ilə ayrılaraq yazılır. Dövrün hissələri isə, öz növbəsində ';' ilə ayrılır. Yəni bu dövrdə də üç hissə var:

Başlanğıc hissə: int i = 0, j = 10

Şərt: i < j

İfadə: i++, j—

Qeyd etmək lazımdır ki, şərt hissəsində bir-birlərindən vergül ilə ayrılmış bir neçə bool dəyişən ola bilməz.

### Sonsuz dövr

Sonsuz dövr dedikdə, heç vaxt sonlanmayan, bir əməliyyatı durmadan yerinə yetirən dövr başa düşülür. Qeyd olunduğu kimi, bir dövr, onun şərti doğru olduğu müddətdə yerinə yetirilir. Deməli, dövrün şərtinin həmişə true qiymət alacağına zəmanət versək, bu dövr heç vaxt sonlanmayacaq – sonsuz dövr olacaq:

for(int i = 0; true; i++)

bu deklorasiya üsulu, sonsuz dövrə uyğundur. Bununla belə, C# - da sonsuz dövr yaratmaq üçün for ifadəsinin xüsusi bir sintaktik şəkli mövcuddur:

```
for(; ;)
{
    Əməliyyatlar
}
```

Bu dövrə aid əməliyyat bloku, kompüter işləyənə qədər, proqram icra olunduğu müddətdə sonlanmayacaq.

## while dövr ifadəsi

Dövr kontrol ifadələrindən biri də while ifadəsidir və bu ifadə ən bəsit dövr ifadəsidir. Çünki bu dövrün deklorasiya şəklində dövrü idarəedən dəyişən və s. olmur. Ümumi

sintaktik şəkli aşağıdakı kimidir:

```
while(şərt)

{

    Əməliyyatlar

}
```

Burada **şərt** bool tipdə bir qiymət alan ifadədir. Əgər şərtin qiyməti true olarsa onda dövr yerinə yetirilir. Beləliklə, əvvəlcə şərt yoxlanılır, əgər doğrudursa dövrün gövdəsi icra olunur, sonra şərt yenidən yoxlanılır, əgər şərt yenə doğrudursa dövrün gövdəsi yenidən icra olunur taki şərt yanlış olana qədər. Yəni bu dövr, ”nə qədər ki, doğrudur, icra et...” prinsipinə dayanır. Əvvəlki cümləni yenidən diqqətlə oxuyun, while ifadəsinin gövdəsi icra olunmamışdan qabaq şərt yoxlanılır. Bu o deməkdir ki, şərt elə ilk başdan yanlış olarsa, dövrün gövdəsi bir dəfə də olsun icra olunmayacaq. İndi bir proqram nümunəsinə baxaq. Bu proqramda daxil edilən ədədin rəqəmlərinin sayını tapacaq:

```
using System;
class soft
{
   static void Main()
   {

      int eded = Convert.ToInt32(Console.ReadLine());
      int say = 0;
      while (eded > 0)
      {
         eded = (eded / 10);
         say++;
      }
      Console.WriteLine("Reqemlerin sayi: " + say);
      Console.ReadKey();
   }
}
```

Nəticə aşağıdakı kimi olur:

```
884476

Reqemlerin sayi: 6
```

Deməli, bu proqramda daxil edilən eded dəyişəninin mərtəbə sayını dövrün içərisində hər dəfə bir vahid azaltdıq və say dəyişəninin qiymətini də bir vahid artırdıq. Bu əməliyyatı eded > 0 şərti daxilində etdik, beləliklə eded > 0 şərti daxilində, daxil edilən ədədin mərtəbə sayı, həmin ədədin rəqəmlərinin sayı qədər azaldıla bilər.

## do-while dövr ifadəsi

while ifadəsinə alternativ olan növbəti dövr kontrol ifadələrindən biri də do- while ifadəsidir. Bu ifadə da eyniylə while kimi işləyir, lakin ondan kiçik bir fərqi var. Qeyd olunduğu kimi, while ifadəsində əvvəlcə şərt yoxlanılır, sonra dövrün gövdəsi icra olunurdu. Yəni, şərtin ilk başdan false olması, while dövrünün heç bir addımının yerinə yetirilməməsinə səbəb olurdu. do- while ifadəsində isə, şərt ilk başdan yoxlanılmır. Bunun yerinə, dövrün gövdəsi bir dəfə icra olunur sonra şərt yoxlanılır. Beləliklə, do-while dövr ifadəsində heç olmasa bir dövrü icra olunur. Yəni, do- while ifadəsinin şərti ilk başdan yanlış olsa belə, bir əməliyyat hökmən yerinə yetirilir. do- while dövr ifadəsinin sintaktik şəkli aşağıdakı kimidir:

```
do

{

    Əməliyyatlar…

}

while(şərt);
```

Minimum bir əməliyyatın icra olunmasının zəruri olduğu əqamlarda do- while dövr ifadəsindən istifadə oluna bilər. İndi bir proqram nümunəsinə baxaq. Bu proqramda daxil edilən ədədin rəqəmlərinin tərsinə düzülmüş variantını alacağıq:

```
using System;
class soft
{
   static void Main()
   {
      Console.WriteLine("Eded daxil edin:");
      int eded = Convert.ToInt32(Console.ReadLine());
      int reqem;
      while (eded > 0)
      {
         reqem = eded % 10;
         eded = eded / 10;
         Console.Write(reqem);
```

```
        }
        Console.ReadKey();
    }
}
```

Deməli, əvvəlcə reqem dəyişəninə daxil edilən ədədin sonuncu rəqəmi mənimsədildi və ekranda əks olundu. Sonra eded dəyişəni bir mərtəbə azaldıldı. Beləliklə, hər dövrdə reqem dəyişəni eded – in axırdan əvvələ doğru rəqəmlərini özündə saxlayacaq. Ekran nəticəsi aşağıdakı kimidir:

```
Eded daxil edin:

123456

654321
```

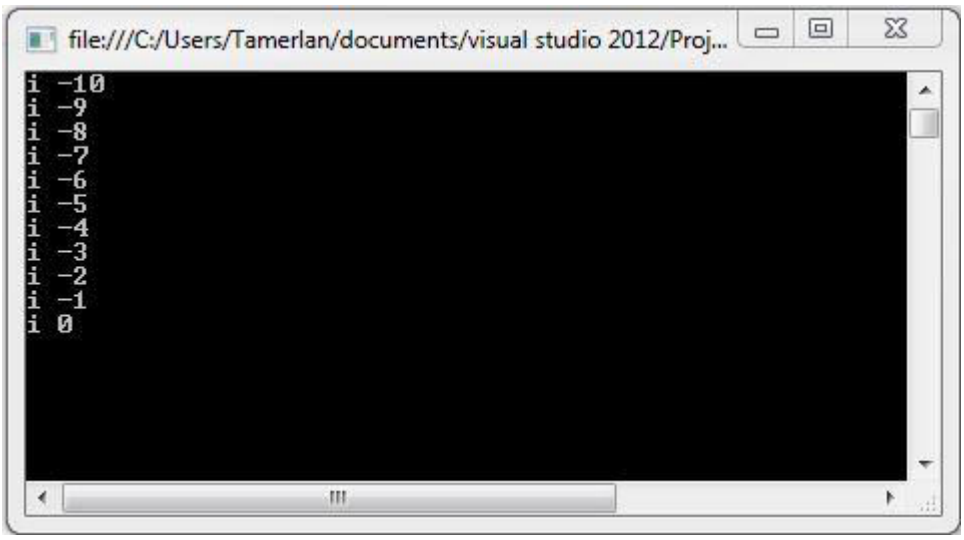
## break ifadəsi

C# - da mövcud kontol ifadələrində biri də break- dır. Bu ifadə, hər hansısa bir dövrü saxlamaq üçün istifadə olunur. break ifadəsi hansısa dövr kontrol ifadəsinin içərisində (gövdəsində) yazılır. Dövrün gövdəsi, break – a çatanda dövrün işi bu nöqtədə sonlandırılır. Bir nümunəyə baxaq:

```
using System;
class soft
{
    static void Main()
    {

        for (int i = -10; i < 10; i++)
        {
            if (i > 0) break;
            Console.WriteLine("i " + i);
        }
        Console.ReadKey();
    }
}
```

Bu proqramın ekran nəticəsi aşağıdakı kimidir:




Deməli, bu proqramda -10 dan 10 – a kimi dövr quruldu və hər iterasiyada dövr kontrol ifadəsinə aid dəyişənin qiyməti ekranda əks olundu. Burada

```
if (i > 0) break;
```

sətrinə fikir verin. Bu ifadənin mənası o deməkdir ki, i dəyişənin qiyməti müsbət olduqda dövrü saxa. Beləliklə, dövr [-10, 9] parçasında icra olunmaqdansa [-10, 0] parçasında icra olundu. İndi daha məqsədəuyğun bir proqrama baxaq. Bu proqramda daxil edilən ədədin ən böyük bölənini tapacaq:

```
using System;
class soft
{
   static void Main()
   {
      Console.WriteLine("Eded daxil edin: ");
      int eded = Convert.ToInt32(Console.ReadLine());
      int eb;
      for (eb = eded / 2; eb > 1; eb--)
      {
         if (eded % eb == 0)
         {
            break;
         }
      }
      Console.WriteLine("En boyuk bolen: " + eb);
      Console.ReadKey();
   }
}
```

Bu proqramda, eb dəyişəni daxil edilən ədədin ən böyük bölənini özündə saxlayacaq. Daxil edilən ədədin yarısından 1 - ə qədər dövr quruldu və hər iterasiya prosesində eded dəyişəninin eb- ə tam bölünməsi yoxlanıldı. İlk uyğunluqda dövr saxlanıldı və əldə olunan eb ədədi, eded dəyişənin ən böyük böləni olacaq. Çünki, dövr kontrol dəyişəninin qiyməti getdikcə azalır. Beləliklə, ekran nəticəsi aşağıdakı kimidir:

```
Eded daxil edin:

38

En boyuk bolen: 19
```

## continue ifadəsi

Dövrlərlə işləmək üçün digər bir kontrol ifadəsi də **continue** – dir. Qeyd olunduğu kimi, break ifadəsi dövrü birdəfəlik saxlayırdı. Yəni, dövrün geri qalan addımlarının sayının nə olmasından asılı olmayaraq, dövr break- a çatanda, dayandırılırdı. break – dan fərqli olaraq, continue ifadəsi dövrü tamam saxlamır. Bunun yerinə, dövrün icrası continue sətrinə çatdıqda cari əməliyyat pas keçilir – ötürülür və iterasiya növbəti addımdan icra olunmağa davam edir. Bir nümunəyə baxaq:

```
using System;
class soft
{
   static void Main()
   {
      for (int i = -10; i < 10; i++)
      {
         if (i == 0) continue;
         Console.WriteLine("i = " + i);
      }
      Console.ReadKey();
   }
}
```

Bu proqramda, i dəyişəninin qiymətinin 0 olduğu vəziyyətdə dövrün işi pas keçilir. Beləliklə, i = 0 olduqda, proqramda continue ifadəsindən dövrün qapanış mötərizəsinə qədərki aradakı kodların heç biri icra olunmur.

# BÖLMƏ 5. OBYEKT YÖNÜMLÜ POQRAMLAŞDIRMA, SİNİFLƏRƏ, OBYEKTLƏRƏ GİRİŞ

## Obyekt yünümlü anlayışı.

Əvvəlcə bu anlayışı proqramlaşdırmadan bağımsız izah etməyə çalışaq. Bir iş görərkən, o işi hissələrə bölmək, o hissələri ayrı-ayrı yerinə yetirmək, sonra hissələr arasındakı məntiqi əlaqəni qurarq yekun işin ortaya çıxarılmasına üsuluna obyekt yönümlü deyilir. Məsələn, bir ticarət mərkəzi düşünün. Ticarət mərkəzlərinə daxil olanda, müxtəlif satış bölmələrinə rast gəlirik: kitab bölməsi, ət məhsullarının satışı bölməsi ya da süd məhsullarının satışı bölməsi. Hər halda kitab satışı bölməsində ət satışını həyata keçirmək çox mənasız olardı. Çünki, bu obyektlər bir-birlərindən asılı deyil. Buna görə də, bir-birlərinə birbaşa aid olmayan obyektləri (kitab, ət, süd, meyvələr) ayrı-ayrı satış bölmələrində yerləşdirmək, daha gözəl üsul olardı. Əgər ət və kitab eyni bir bölmədə satılsaydı, bu vəziyyətdən nə satıcı, nə də müştəri razı qalardı. Hətta bu vəziyyəti görən müştəri böyük ehtimalla bir daha o ticarət mərkəzinə getməyəcəkdir. Beləliklə, bir-birlərinə dəxli olmayan obyektləri ayıraraq, biz obyekt yönümlü satışı həyata keçirdik. Bu prinsipə söykənərək proqram yazmağa imkan verən proqramlaşdırma dillərinə də obeyekt yönümlü proqramlaşdırma dili (object oriented programming language) deyilir. Obyekt yönümlü proqralaşdırma dillərinə bu qədər bəsit və primitiv tərif vermək, əslində o qədər də düzgün deyil. Obyekt yönümlülüyü təmin edən bəzi kriteriyalar var ki, bölmənin davamında qeyd olunacaq. Bununla belə, OYP dillərində obyekt yönümlülüyün əsas vahidi sinifdir. Yəni, bir-birlərinə birbaşa bağlı olmayan proqram kodlarını ayrı-ayrı siniflərdə yazaraq obyekt yönümlü proqram yazmış oluruq. Bir proqram yazarkən, əvvəlcə həmin proqramı yaxşıca analiz edirik və proqramımızda mövcud olacaq proqram modullarını bir-birlərinə qarışdırmadan ayrı-ayrı siniflərdə yazırıq. Obyekt yönümlü olmadan böyük həcmli proqramlar yazmaq çox çətin idi, hətta mümkün deyildi. Çünki, məsələn 5000 sətir kod yazdıqdan sonra proqramı idarə etmək mümkünsüz hala gəlirdi. Bu, böyük bir spagetti boşqabının içindən hansısa vermişel dənəsini axtarmağa çalışmaq kimi idi. Obyekt yönümlü proqram yazmaq ona görə vacibdir ki, böyük proqramları rahatlıqla yaza bilər, bir aydan sonra proqramımıza baxanda həmin proqramı tanıya bilək. Dinamik proqramlar yaza bilək (update oluna bilən). Bu səbəblər uzanır gedir. Amma deyilənləri ümumiləşdirsək, obyekt yönümlü olaraq proqram yazmaq ona görə əlverişlidir ki, bu zaman kompüter kimi "düşünməyə" ehtiac yoxdur. Adi həyatda işlərinizi necə edirsinizsə, eyni düşüncə və məntiqlə rahatlıqla proqramlarınızı obyekt yönümlü olaraq yaza bilərsiniz. Nə yaxşı ki, C# bizə tamamilə obyekt yönümlü proqram yazmağa imkan verir. Yaşasın C#!

Obyekt yönümlü proqramlaşdırma dillərində özünü göstərən 4 əsas faktor var:

- Abstraksiya / Mücərrədlik : Abstraction
- İnkapsulyasiya : Encapsulation
- Polimorfizm: Polymorphism

● Varislik : Inheritance
Beləliklə, bir proqramlaşdırma dillərində yuxarıda göstərilən xüsusiyyətlərin hamısı dəstəklənirsə, o artıq tamamilə obyekt yönümlü (pure object oriented) proqramlaşdırma dili olur, hansı ki C# bu sinfə aiddir.

## Abstraksiya (Abstraction)

Əvvəlcədən qeyd edim ki, obyekt yönümlü proqramlaşdırma (OYP) dillərində bir anlayışı bilmədən digərini tam başa düşmək mükün deyil. Çünki, bir anlayış digəri ilə sıx bağlıdır. Ona görə də qeyd olunanları, tam olaraq başa düşməməyiniz normaldır.

Abstraksiya, bir obyektin müəyyən hissələrini kənar dünyadan xaric etmə əməliyyatıdır. Yəni obyektin, daxili mexanizmini yox, sadəcə funksionallığının təqdim olunmasıdır. Məsələn, böyük ehtimalla mobil telefon istifadə edirsiniz, yəni sadəcə istifadə edirsiniz. Yəni, cihazın daxilində gedən çox mürəkkəb "digital" proseslər sizi maraqlandırmır. Siz sadəcə, obyektin (telefon) kənar dünyaya təqdim olunan hissələri ilə işləyirsiniz. Ya da bir paltaryuyan maşının içərisində gedən proseslər insanlar üçün maraqlı deyil. Bütün o proseslər kombinasiyası insanlara bir neçə düymə ilə təqdim olunur. Beləliklə, bir müvafiq düymələri sıxmaqla obyektdən istifadə edirik. Proqramlaşdırma nöqteyi nəzərdən baxsaq, abstraksiya bir obyektin necə işləməyi ilə maraqlanmayıb, onu sadəcə istifadə etməkdən ibarətdir. Məsələn, bir fikirləşin, ekranda bir şey çap etdirəndə sadəcə

```
Console.WriteLine()
```

deyə bir ifadə yazırıq. Əslində pərdə arxasında ekranda əks olunacaq söz, çevrilir binar formata, müfaviq kodlaşdırmalar tənzimlənir, proqram məlumatı sistemə göndərir, sistem kompüterə qoşulmuş ekran kartlarını analiz edir və hazırda işlək vəziyyətdə olan ekran kartına binar formatdakı məlumatı göndərir. Ekran kartı da alınış məlumatı işləyərək, özünə qoşulmuş monitorun müvafiq piksellərini yandırır bla bla bla... Yəni, obyektin bizə necə işləməyi yox, sadəcə nə işə yaradığının məlum olmağı gərəkdiyi məqamlarda abstraksiya anlayışı özünü göstərir. Yox əgər siz özünüzü mühəndis hesab edirsinizsə, onda keçin C++ - a...

## İnkapsulyasiya (Encapsulation)

Bu anlayış, abstraksiya anlayışına yaxın bir anlayışdır. İnkapsulyasiya dedikdə, bir obyektin müəyyən hissələrini istifadəçilərdən gizlətmək başa düşülür. Beləliklə, İnkapsulyasiya, bir bir sinfin proqramçını maraqlandırmayan hansısa üzvlərini ondan gizlətmək ya da bir sinfin üzvlərinə kənar dünyadan müdaxilə edilməsinin qarşısını almaq məqsədilə instifadə oluna bilər. Paltaryuyan məsələsinə qayıtsaq, əgər, həvəskar

şəkildə paltaryuyan maşının içini açsaq, ya onu yararsız vəziyyətə salarıq ya da elektrik vurmasından xəsarət alarıq. Yəni, bir obyektin hissələrinə kənardan nə cür müdaxilə olunabiləcəyini təyin etməkdir – inkapsulyasiya. C# - da inkapsulyasiya əməliyyatı üçün "Hüquq təyinedicilər" (Access Modifiers) adlanan açar sözlərdən istifadə olunur.

## Hüquq təyinedicilər (Access Modifiers)

Hüquq təyinedicilər, inkapsulyasiyanın göstəricisi olub, istifadəçilərin bir sinif içərisindəki dəyişənlərdən və metodlardan nə cür və ya nə zaman istifadə edə biləcəklərini təyin edən attributlardır. Başqa sözlə desək, bir sinfin üzvlərini sadəcə təyin olunmuş sahələrdə istifadə etəyə icazə verən xüsusiyyətlərdir. Mövzunu tam olaraq anlamaya bilərsiniz çünki, sizin sinif anlayışınız yoxdur. Obyekt yönümlü dillərdə bir sinfin üzvərindəki istifadə hüquqları əsas iki istiqamətdə təsvir olunur: Hamıya açıq (public) və Özəl (private). Bir sinfin hansısa üzvünü public olaraq təyin etməklə, o üzvü proqramın hər yerində istifadə etməyə icazə veririk. Bir sinfin üzvlərini private olaraq təyin etmək, həmin üzvü sadəcə sözügedən sinfin içərisində istifadəyə etməyə icazə vermək deməkdir. Bir sinfin üzvlərinin görüləbilənliyini təyin edən 4 təyinedici mövcuddur:

1. public – üzvlərə proqram daxilində hər yerdən müraciət oluna bilər

2. private – üzvlərə sadəcə yerləşdiyi siniflərdən müraciət oluna bilər, kənar siniflərdən müraciət hüququ yoxdur

3. protected – üzvlərə kənar sinif olaraq, sadəcə müvafiq sinifdən törəyən siniflər daxilində müraciət oluna bilər (Varislik bölməsində qeyd olunacaq)

4. internal – üzvlərə eyni assembler daxilində müraciət oluna bilər (Reflection API bölməsində qeyd olunacaq).

## Siniflərin əsasları

C# demək sinif deməkdir. Qeyd olunduğu kimi, sinif obyekt yönümlü proqramlaşdırmanın əsas vahididir. Bu xüsusilə də C# - da belədir. Ticarət mərkəzi məsələsinə qayıtsaq, hər satış bölməsini bir sinif kimi təsəvvür edə bilərik. Siniflərin içərisində də həmin sinfə aid üzvlər olur. Kitablar, kitablar haqqında məlumatlar, kitab satıcısı həmin kitab satışı sinfinin üzvləridir. C# - da da sinfin üzvləri dedikdə metodlar,

örnək dəyişənlər, sabitlər, xüsusiyyətlər, indeksləyicilər, konstruktorlar və s. başa düşülür. Sonuncu cümləni başa düşmədiyinizi bilirəm. Bu deyilənlər, müvafiq bölmələrdə qeyd olunacaq. C# tamamilə obyekt yönümlü bir dildir, buna görə də hər bir fəaliyyət siniflərin içərisində meydana gəlməlidir. Yəni, hər bir C# proqramında ən azı bir sinif mövcud olmalıdır, siniflərdən kənarda heç bir şey ola bilməz. Necə ki, ticarət mərkəzində ən azı bir satış bölməsi olmalıdır. Yoxsa, həmin məkan ticarət mərkəzi olmazdı. Fikir versəniz, indiyə qədərki bütün proqram nümunələrində heç olmasa bir sinif istifadə etmişdik. Əslində siz bunun fərqində olmamısınız.

Hər şeyin siniflər içərisində olduğunu qeyd etdik. Buna görə də əvvəlcə bir sinif daxil edirik, sonra da həmin sinfin içərisində icra olunacaq kodları qeyd edirik. Bir sinif təyin etmək üçün aşağıdakı ümumi sintaksistən istifadə edəcəyik:

**class** sinfin-adı

{

Üzvlər...

}

Burada **class** C# - ın açar sözlərindən biridir və bir sinif təyin etmək üçün istifadə olunur. Aşağıdakı proqrama baxaq:

```
using System;
class TicaretM
{
   static void Main()
   {
      Console.WriteLine("Salam, dunya!");
   }
}
```

Bu proqramda bir ədəd "TicaretM" adlı sinif var. Bu sinfin içərisində isə Main() metodu var. Yəni bu sinfin bir dənə üzvu var ki, o da Main() metodudur. Siniflərin üzvləri həmin sinfin içərisində yerləşən örnək dəyişənlər, metodlar, xüsusiyyətlər, indeksləyicilər və s ola bilər. Bir sinif təyin etmək, bir verilənlər tipi yaratmaq deməkdir. Yəni, yuxarıdakı proqramda bir "TicaretM" adlı sinif yaratmaqla həm də eyni adlı bir verilənlər tipi də yaratmış olduq. İndi verilənlər tipləri mövzusunu yadınıza salsanız, qeyd etmişdim ki, tilər iki yerə bölünürdü: dəyər tipləri (valuable types) və referans tipləri (reference types). Dəyər tipləri C# - ın standart tipləridir. Referans tipləri isə siniflərdir. Yəni, biz bu proqramda referans tipi kateqoriyasında bir verilənlər tipi yaratdıq.

## Örnək dəyişənlər (Instance variables)

Örnək dəyişənlər dedikdə, bir sinif içərisində təyin olunmuş və o sinif daxilində hər yerdə "görünən" qlobal dəyişənlərdir. "Verilənlər tipləri, dəyişənlər" bölməsində qeyd olunduğu kimi, dəyişənləri iki hissəyə ayırmışdıq: Qlobal dəyişənlər və lokal dəyişənlər. Bax qlobal dəyişənlər həmin bu örnək dəyişənlərdir ki, bu dəyişənlərə də sahələr (fields) deyilir. Lokal dəyişənlərdən fərqli olaraq örnək dəyişənlər, bir kod bloku üçün yox, bir sinif və ya bütöv proqram üçün təyin olunan ümumi dəyişənlərdir. Örnək dəyişənləri də adi dəyişənlər kimi təyin edirik, tək fərqi, örnək dəyişənlərə proqramın hansı hissəsindən müraciət olunduğunu təyin etmək üçün, həmin dəyişənləri üçün hüquq təyinedici ilə birlikdə yaratmalıyıq. Örnək dəyişənlərin ümumi deklorasiya şəkli aşağıdakı kimidir:

hüquq-təyinedici tip dəyişənin adı;

Məsələn aşağıdakı proqramda TicaretM sinfi içərisində iki ədəd "mudir" və "satici" adlı dəyişənlər təyin olunub:

```
class TicaretM
{
   private string mudir;
   public string satici;
}
```

Fikir verin! Burada mudir dəyişəni public, satici dəyişəni isə private olaraq təyin olunub. Yəni, bu sinfin satici adlı dəyişəninə proqramın hər yerində müraciət oluna biləcəyi halda, mudir dəyişəninə sadəcə TicaretM sinfi daxilində müraciət oluna bilər. Dərindən nəfəs alın və narahat olmayın. Çünki, mövzunu tam olaraq başa düşmədiyinizi bilirəm. Haqlısınız, mən də sizin yerinizdə olsaydım, "Bu Tamerlan nə yazıb görəsən?" deyərdim. Bütün sistem beyninizdə oturacaq, amma hələ yox. Deməli, mudir adlı dəyişəni kənar sinifdən "görə" bilmərik. Bunu isbat etmək üçün ikinci bir sinif yaradaq və bu sinif daxilində TicaretM adlı sinfin mudir adlı üzvünə qiymət mənimsətməyə çalışaq. Amma bir şey var... Başqa bir sinfin içərisindəki üzvlərə müraciət etmək üçün əvvləcə həmin sinfə aid bir obyekt yaratmalıyıq. Qarşımıza yeni bir anlayış çıxdı, obyekt...

## Obyektlərin təyini

Ümumiyyətlə obyekt dedikdə, yaddaş sahəsinə malik ya da yaddaşda müəyyən yer tutan hər şey başa düşülür. Bir sinfin üzvlərinə müraciət etmək üçün, bizim də bir yaddaş sahəsinə ehtiyacımız var, hansı ki, bu yaddaş sahəsi müvafiq sinfin elementlərini özündə saxlaya biləcək formatda olsun. Yəni, yaddaş sahəsi ayıracağıq – obyekt yaradacağıq və həmin obyekt vasitəsi ilə siniflər içərisindəki üzvlərə müraciət edəcəyik. Bir sinfə aid bir obyekt yaratmaq üçün aşağıdakı sntaksisdən istifadə etmək olar:

sinfin-adı obyekt = new sinfin-adı();

Burada **sinfin-adı** obyektini yaratmaq istədiyimiz sinfin adıdır. Aşağıdakı proqrama baxaq. Bu proqramda iki ədəd "TicaretM" və "Program" adlı sinif olacaq. Program sinfi içərisindəki Main() metodu daxilində TicaretM sifinin elementlərinə müraciət edəcəyik:

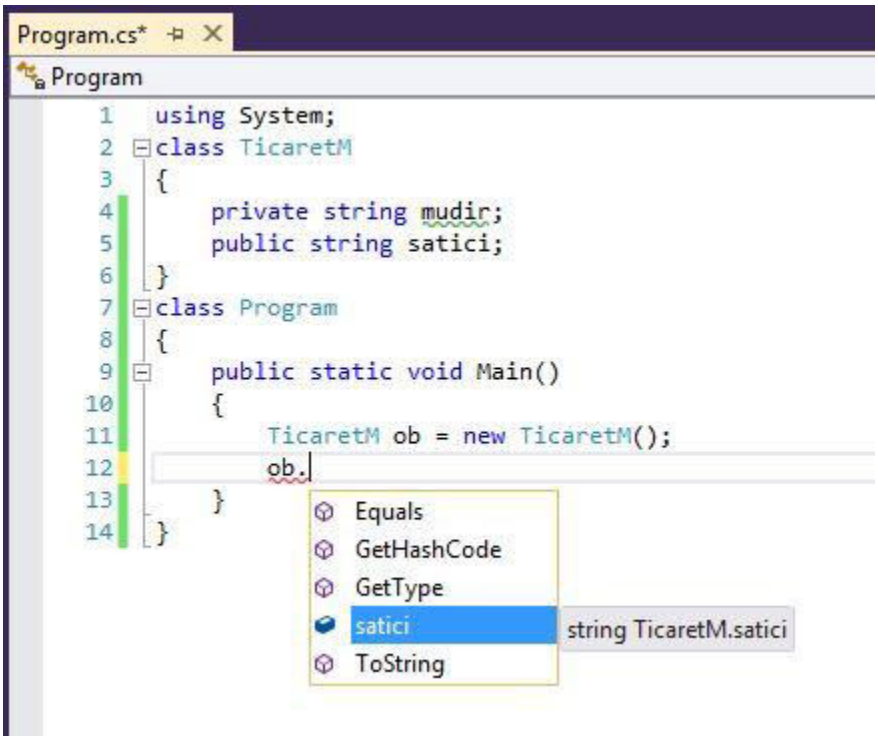
```
using System;
class TicaretM
{
   private string mudir;
   public string satici;
}

class Program
{
   public static void Main()
   {
      TicaretM ob = new TicaretM(); //(1)
      ob.satici = "Sadiq Memmedov";
      Console.WriteLine("ob obyekti ucun satici: " + ob.satici);
      Console.ReadKey();
   }
}
```

Program sinfindəki (1) sətrinə fikir verin. Bu kod sətri ilə TicaretM ainsinə aid bir obyekt (yaddaş sahəsi) ayırdıq və həmin obyektlə TicaretM sinfinin "satici" dəyişəninə qiymət mənimsətdik. Beləliklə, ekran nəticəsi aşağıdakı kimi olur:

```
ob obyekti ucun satici: Sadiq Memmedov
```

İndi, çox qarışdırmadan əsas fikrə keçək. TicaretM sinfindəki mudir adlı dəyişənin hüquq təyinedicisi göründüyü kimi private – dir. Bu o deməkdir ki, bu dəyişənə kənar siniflər içərisindən müraciət oluna bilməz. Doğrudan da "ob" adlı obyekti yazıb sonra '.' qoyduqdan sonra Visual Studio IDE bizə IntelliSense pəncərəsində sadəcə TicaretM sinfinin, ob obyektinin hüququ çatan üzvlərini göstərdi:




Göründüyü kimi, siyahıda mudir dəyişəni görünmür. Əgər private olduğu halda ob.mudir ifadəsi ilə bu üzvə müraciət etmək istəsək, "'TicaretM.mudir' is inaccessible due to its protection level" sintaksik xətasını alarıq. Obyektlər haqqındakı bu açıqlamaların sizi qane etmədiyini bilirəm, qane etsə də, obyektlər haqqında bilməli oduğumuz vacib şeylər var.

## Obyektlər necə işləyir?

Yuxarıda TicaretM sinfinə aid bir obyekt yaradanda

```
TicaretM ob = new TicaretM();
```

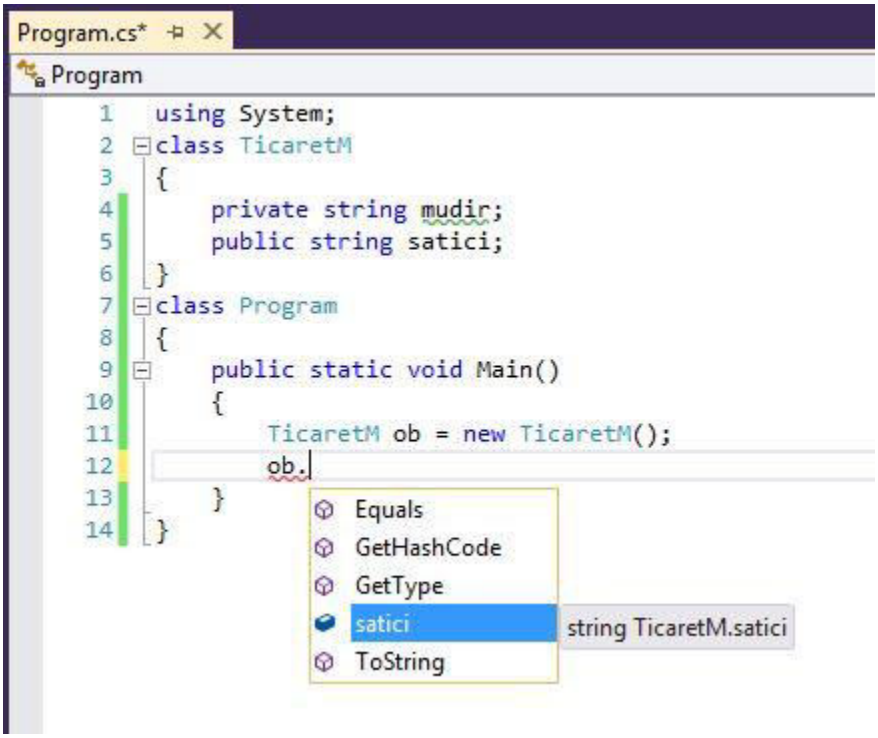
kimi bir sətirdən istifadə etdik. Əslində, bu sətri aşağıdakı kimi iki hissədən ibarət yaza bilərik:

```
TicaretM ob;

ob = new TicaretM();
```

Siniflərin əslində bir verilənlər tipi olduğunu demişdik. Beləliklə, birinci sətirdə TicaretM tipinə aid ob adlı bir **referans** dəyişən təyin etdik. Bu hələ obyekt deyil, çünki yaddaşı yoxdur! Bu vəziyyətdə dəyişənin qiyməti hələki NULL – dur. İkinci sətirdə isə bu referans dəyişəni bir yaddaş sahəsi ilə əlaqələndirdik və artıq obyektimizi yaratmış olduq. Bu sətirdə **new** opertoruna fikir verin. Bu operator referans dəyişənləri ilə işlədildikdə, həmin dəyişənləri dinamik olaraq (yəni işləmə zamanı) bir yaddaş sahəsinə bağlayır.

Əgər bir referans dəyişənini hansısa yaddaşla əlaqələndirməmiş (Null vəziyyətdə) müvafiq sinfin elementlərinə müraciət etmək istəsək, "NullReferenceException" xətasını alarıq. Aşağıdakı proqram düzgün deyil:

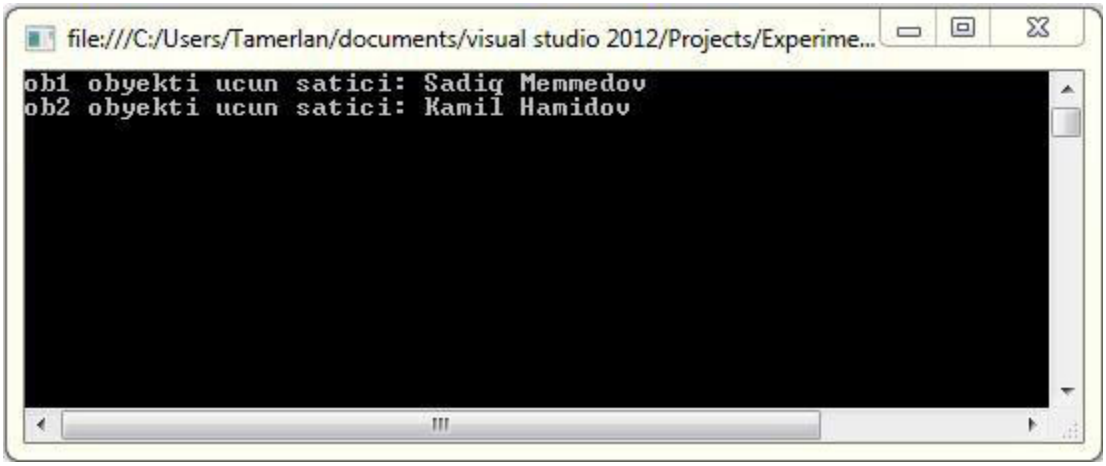



Bir obyekt yaradanda, həmin obyekt aid olduğu sinfin üzvlərinin kopyasını (hüququnun çata biləcəyi) öz yaddaş sahəsində saxlayır. Yəni, hər obyekt, aid olduğu sininf elementlərinin ayrı bir kopyasını özündə saxlayır. Orijinal elementlərdə bir dəyişiklik olmur. Buna görə də bir obyekt aid elementlər üzərində edilən dəyişikliklər, digər obyektlərin elementlərinə təsir etmir. Bunu göstərən aşağıdakı proqrama baxaq:

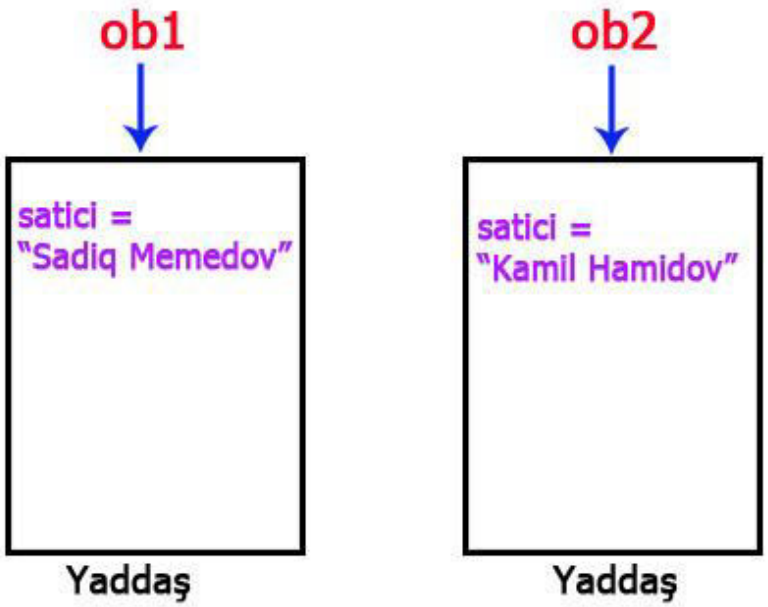
```
using System;
class TicaretM
{
   private string mudir;
   public string satici;
}
class Program
{
   public static void Main()
   {
      TicaretM ob1 = new TicaretM();
      TicaretM ob2 = new TicaretM();
      ob1.satici = "Sadiq Memmedov";
      ob2.satici = "Kamil Hamidov";
```

```
        Console.WriteLine("ob1 obyekti ucun satici: " + ob1.satici);
        Console.WriteLine("ob2 obyekti ucun satici: " + ob2.satici);
        Console.ReadKey();
    }
}
```

Bu proqramın ekran nəticəsi aşağıdakı kimi olur:




ob1 obyektinə aid yaddaş sahəsindəki elementər üzərində edilən dəyişikiklər, ob2 obyektinin eleentlərinə təsir etmir. Bunu aşağıdakı qrafik çox gözəl təsvir edir:

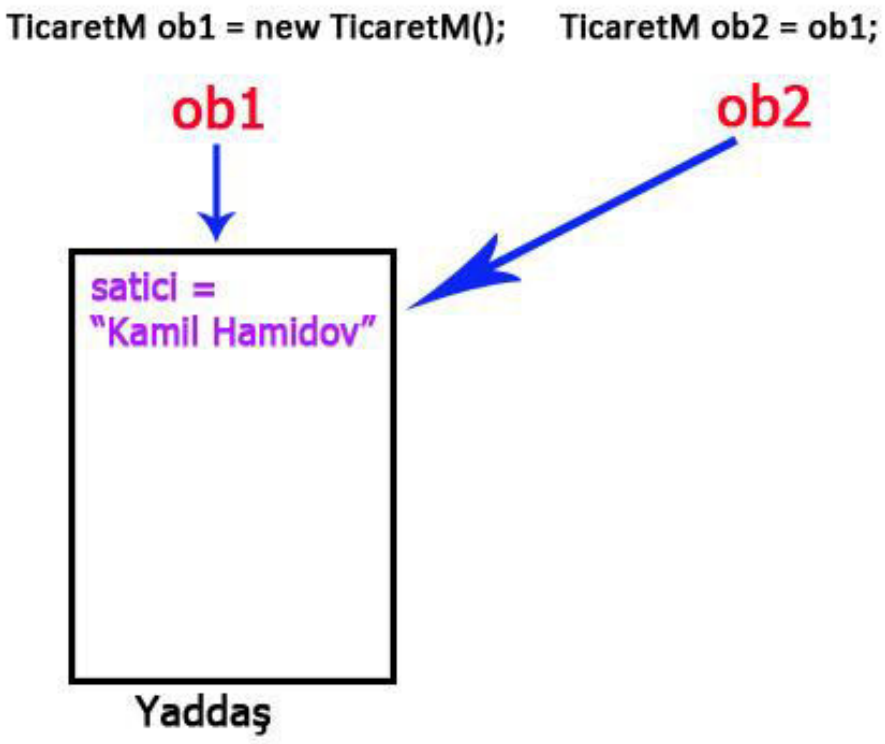


## Referans dəyişənləri üzərində mənimsətmə

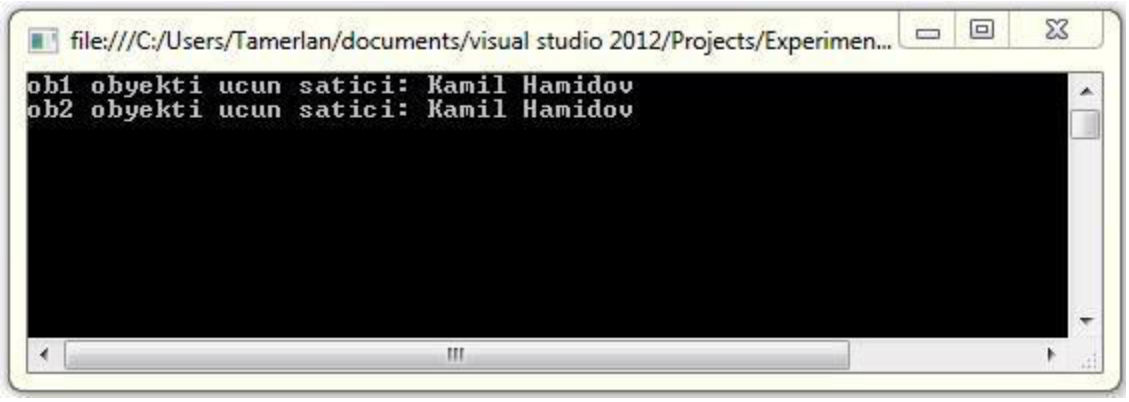
Referans tiplərinə aid dəyişənlərlə dəyər tiplərinə aid dəyişənlər üzərində mənimsətmə əməliyyatı ciddi fərqlilik göstərir. C# nöqteyi nəzərdən bunu başa düşmək vacibdir. Bir dəyər tipindəki dəyişən, ona məsimsədilən qiyməti özü bilə birlikdə daşıyır. Amma referans dəişənlər üçün bu belə deyil. Referans dəyişənlər, məlumatları birbaşa özlərində daşımır, bunun əvəzinə məlumatların saxlanıldığı yaddaşın ünvanını özlərində saxlayır. Buna görə də, məsələn iki referans dəyişən eyni bir yaddaş sahəsinə referans edirsə (istinad edirsə) onda bu referans dəyişənlərinin biri vasitəsi ilə yaddaşda edilən dəyişiklik, digəri üçün da keçərli olacaqdır. Aşağıdakı proqram bu faktı çox gözəl əks etdirir:

```
using System;
class TicaretM
{
   private string mudir;
   public string satici;
}
class Program
{
   public static void Main()
   {
      TicaretM ob1 = new TicaretM(); //(1)
      TicaretM ob2 = ob1;            //(2)
      ob1.satici = "Sadiq Memmedov";
      ob2.satici = "Kamil Hamidov";
      Console.WriteLine("ob1 obyekti ucun satici: " + ob1.satici);
      Console.WriteLine("ob2 obyekti ucun satici: " + ob2.satici);
      Console.ReadKey();
   }
}
```

(1) sətrində ob1 adlı referans dəyişənini müəyyən bir yaddaş sahəsinə bağladıq. (2) sətrində isə ob2 adlı yeni referans dəyişənə ob1 dəyişənini mənimsətdik. Beləliklə, ob2 dəyişəni ob1 ilə eyni adresə istinad edəcəkdir. Aşağıdakı qrafik bunu gözəl təsvir edir:




Beləliklə, proqramın nəticəsi aşağıdakı kimi olur:



# BÖLMƏ 6. METODLAR

## Metod anlayışı

Metodlar bir sinfin əsas üzvlərindən biridir və C#- ın ən vacib, o cümlədən, fundamental anlayışları sırasındadır. Ümumiyyətlə metod, xaricdən bir qiymət alan (ya da almayan), müəyyə əməliyyatları yerinə yetirən və geriyə bir nəticə qaytaran (ya da qaytarmayan) kod bloklarıdır. Metodlar, birmənalı olaraq proqram kodlarının gərəksiz yerə təkrarlanmasının qarşısı alınır. Ticarət mərkəzi məsələsinə qayıtsaq, müştəri müxtəlif bölmələrdən alış-veriş edir. Hər bölmədən alver edib hər bölməyə də lazımi qədər pul ödmək əslində o qədər də yaxşı deyil. Bunun yerinə, müştəri istədiklərini müxtəlif bölmələrdən alıb və çıxışda yekun məbləği bir dəfəlik kassaya ödəməsi, daha gözəl bir üsuldur. Onsuz da, pul ödəmək eyni bir əməliyyatdır, bunu hər bölmə üçün təkrarlamaq mənasızdır. Bax metodlar da eyni əməliyyatların (kodların) dəfələrlə təkrarlanmasının qarşısını almaq üçün yaradılıb. Lazımi kodları bir metod kimi təyin etmək və istədiyimiz vaxt həmin metodu **çağırmaq** kifayətdir. Metodlar bir sinfin üzvləridir. Qeyd olunduğu kimi, metodlar müəyyən bir əməliyyat yerinə yetirir və nəticə olaraq geriyə bir məlumat qaytara ya da qaytarmaya bilər. Əgər heç bir məlumat qaytarmasa, onda həmin metodların qaytarıma tipi **void** olmalıdır. Metodlar kənardan bir məlumat alaraq o məlumatlar əsasında müəyyən əməliyyatlar yerinə yetirə bilər. Belə metodlara isə **parametrli** petorlar deyilir. Beləliklə, metodları aşağdakı qayda ilə təyin edəcəyik:

```
<hüquq-təyinedici> <qaytarılma-tipi> <metodun-adı>(parametrlər)
{
    kodlar…
}
```

Bu təriflər, sizə bir az qarışıq gələ bilər. Amma çətin bir şey yoxdur. İndi ən sadə şəkildə, adi həyat məsələləri üzərində bu deyilənləri başa salmağa çalışacam. Başlayaq, geriyə qiymət qayratmayan metodlardan. Deməli, yazdım ki, bu cür metodlar hansısa işi görür bu işin nəticəsi kimi, geriyə bir məlumat vermir. Sadəcə iş görür. Ticarət mərkəzindəki kassa motodumuz olsun hansı ki, bu metod alınan malların dəyərini toplama əməliyyatını yerinə yetirir. Əgər müştəri aldığı malların dəyərini tam dəqiq şəkildə kassaya ödəyərsə, onda kassa müştərisə heç bir pul qalığı **qaytarmayacaq**. Yəni, kassa metodu geriyə qiymət qatarmayan metod oldu. Əks halda kassa metodu pulu hesablayacaq və geriyə qalığı **qaytaracaq**. Bu isə geriyə bir qiymət qaytaran metod oldu. Müştəri kassaya pul ödəyir, yəni kassa metoduna bir məlumat göndərir (pul) və

kassa metodu da məlumatı qəbul edərək işləyir. Bu zaman kassa metodu **parametrli** metod olur. Pul həmin metodun parametridir. Amma dükanın sahibinin kassaya bir şey ödəməsi gərəkmir. Bu zaman kassa parametrsiz metod olur. Bundan bir az fərqli olaraq, C# - da bir metod eyni anda bu göstəricilərin hamısını özündə saxlaya bilməz. Yəni, eyni bir metod həm qiymət qaytaran həm də qaytarmayan ola bilməz. Ya qaytarır, ya da qaytarmır! Eyniylə, eyni bir metod həm parametri həm də parametrsiz ola bilməz.

## Qiymət qaytarmayan metodlar

Bu başlıq altında ən sadə prototipə malik metodlarə öyrənəcəyik – heç bir qiymət qaytarmayan və parametrsiz metodları. Bu cür metodların sintaksisi aşağıdakı kimidir:

<hüquq-təyinedici> void <metod-adı>()

{

kodlar…

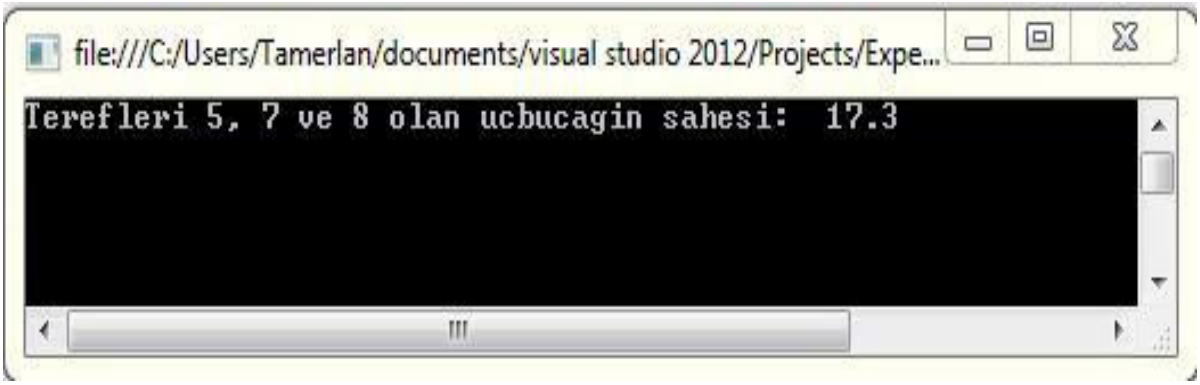
}

Burada **void** metodun geriyə heç bir qiymət qaytarmadığı anlamına gəlir. Aşağıdakı proqramda Ucbucaq sinfi və bu sinif içərisində ixtiyari üçbucağın sahəsini Heron düsturu ilə hesablayan bir proqram təsvir olunub:

```
using System;
class Ucbucaq
{
    public int teref1;
    public int teref2;
    public int teref3;
    public void Sahe()
    {
        int p = (teref1 + teref2 + teref3) / 2;
        double sahe = Math.Sqrt(p * (p - teref1) * (p - teref2) * (p - teref3));
        Console.WriteLine("Terefleri {0}, {1} ve {2} olan ucbucagin sahesi: {3: ##.#}",
teref1, teref2, teref3, sahe);
    }
}
class Program
{
    public static void Main()
    {
        Ucbucaq ob = new Ucbucaq();
```

```
        ob.teref1 = 5;
        ob.teref2 = 7;
        ob.teref3 = 8;
        ob.Sahe(); //(1) Metodu cagir
        Console.ReadKey();
    }
}
```

Bu proqramda iki ədəd sinif var: Ucbucaq və Program. Ucbucaq sinfi içərisində üç ədəd tam tipdə dəyişən təyin etdik və bir ədəd Sahe() adlı metod təyin etdik. Bu metod tərəflərə görə sahəni hesablayacaq. Program sinfinin Main() metodu içərisində bu sinfə aid bir obyekt yaratdıq və bu obyekt üçün teref1, teref2, teref3 dəyişənlərinə uyğun olaraq 5, 7, 8 qiymətlərini verdik. Sonra bu obyekt ilə - (1) sətrinə fikir verin - Sahe() metodunu çağırdıq. Çağırma, obyektin adından sonra nöqtə (.) operatorunun qoyulması ilə həyata keçdi. Metodların təyinində və çağrılma sətirlərində mötərizələrin olmasına diqqət edin. Sahe() metodu isə özünü çağıran obyekt (yaddaş) içərisindəki qiymətlərə əsasən sahəni hesabladı. Deməli, proqram Main() metodunun ilk fiqurlu mötərizəsindən başlayaraq icra olunmağa başladı və (1) sətrində proqramın axış istiqaməti Ucbucaq sinfi içərisindəki Sahe() metoduna yönləndi. Sahe() metodu icra olunduqdan sonra (axış, bu metodun qapanış fiqurlu mötərizəsinə çatdıqda) proqram icrasını (1) sətrindən aşağı doğru davam etditməyə başlayır. Beləliklə, nəticə aşağıdakı kimi olur:




Başa düşmək lazımdır ki, hər metod müəyyən əməliyyatı yerinə yetirərkən, özünü çağıran obyektə (yaddaşa) aid məlumatlardan istifadə edir. Bu proqramda Sahe() metodunu ob obyekti çağırdığı üçün, Sahe() metodu öz gövdəsində teref1, teref2, teref3 dəyişənləri üçün uyğun olaraq 5, 7, 8 qiymətlərindən istifadə etdi. Başqa bir obyektlə bu metodu çağırsaydıq, o zaman bu dəyişənlər üçün istifadə olunan qiymətlər də həmin obyekt üçün teref1, teref2, teref3 dəyişənlərinin qiymətləri olacaqdır. Obyektlərin işləmə prinsipi başlığı atında qeyd etmişdim ki, hər obyekt aid olduğu sinfin üzvlərinin bir kopyasını özündə saxlayır. Nəticə etibarı ilə, obyektlər eyniylə aid olduğu

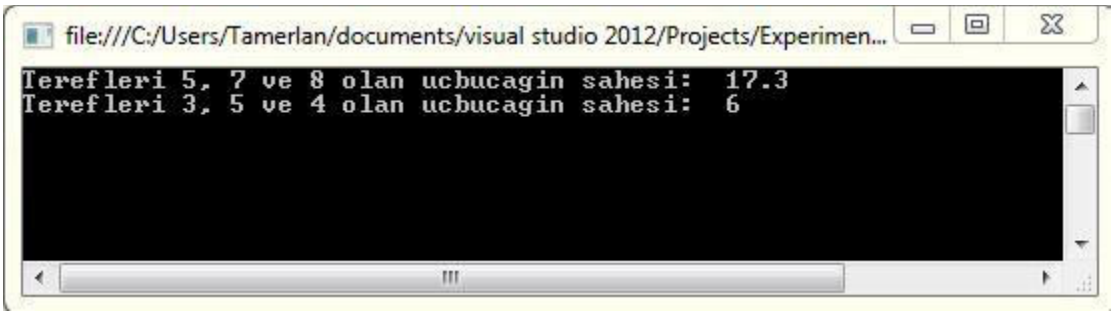
sinfə aid metodların da bir kopyasını özündə saxlayır. Aşağıdakı proqramda nə demək istədiyim, tam olaraq əks olunub:

```
using System;
class Ucbucaq
{
    public int teref1;
    public int teref2;
    public int teref3;
    public void Sahe()
    {
        int p = (teref1 + teref2 + teref3) / 2;
        double sahe = Math.Sqrt(p * (p - teref1) * (p - teref2) * (p - teref3));
        Console.WriteLine("Terefleri {0}, {1} ve {2} olan ucbucagin sahesi: {3: ##.#}",
teref1, teref2, teref3, sahe);
    }
}
class Program
{
    public static void Main()
    {
        Ucbucaq ucbucaq1 = new Ucbucaq();
        ucbucaq1.teref1 = 5;
        ucbucaq1.teref2 = 7;
        ucbucaq1.teref3 = 8;

        Ucbucaq ucbucaq2 = new Ucbucaq();
        ucbucaq2.teref1 = 3;
        ucbucaq2.teref2 = 5;
        ucbucaq2.teref3 = 4;

        ucbucaq1.Sahe();
        ucbucaq2.Sahe();
        Console.ReadKey();
    }
}
```

Bu proqramın nəticəsi aşağıdakı kimi olur:




Yeri gəlmişkən, Ucbucaq sinfi içərisindəki üzvlərin public hüquq təyinedicilərinə malik olduğuna da fikir verin. Yoxsa onlara, Program sinfi içərisindən müraciət edə bilməzdik. Çox yoruldum, sabah davam edərəm...

## Qiymət qaytaran metodlar

Bir metodun qiymət qaytarması dedikdə, metodun müəyyən əməliyyatlar yerinə yetirib o əməliyyatların nəticəsi kimi, bir məlumatın ixrac olunması başa düşülür. Metodlar geriyə hərhansısa tipə aid bir qiymət qaytra bilər. Buna görə də, metodun qaytaracağı qiymətlərin tipi bu metodun qaytarılma tipi adlanır. Metodların qiymət qaytarması, **return** açar sözü ilə həyata keçirilir. Geriyə qiymət qaytaran (prametrsiz) metodların sintaksisi aşağıdakı kimidir:

**<hüquq-təyinedici> <qaytarılma-tipi> <Metod-adı>()**

**{**

**Kodlar...**

**return qiymət;**

**}**

Burada, hüquq-təyinedici metoda ənar sinifəlrdən nə cür müraciət oluna biləcəyini təyin edir (public, private və s.). **qaytarılma-tipi** metodun qaytaracağı qiymətin tipidir. Məsələn, Ucbucaq sinfindəki metod sahəni hesablayırdı və elə öz gövdəsində də bu sahəni ekranda çap edirdi. Bu metodu qiymət qaytaran şəklə optimizasiya edək, belə ki, metod tərəflərə görə sahəni hesablasın və bu qiyməti geriyə qaytarsın. Biz də qaytarılanbu qiyməti əldə edək və bunu Main() metodu içərisində ekranda çap edək:

```
using System;
class Ucbucaq
{
   public int teref1;
```

```
    public int teref2;
    public int teref3;
    public double Sahe()
    {
        int p = (teref1 + teref2 + teref3) / 2;
        double sahe = Math.Sqrt(p * (p - teref1) * (p - teref2) * (p - teref3));
        Console.Write("Terefleri {0}, {1} ve {2} olan ucbucagin sahesi: ", teref1, teref2,
teref3);
        return sahe; //(1)
    }
}
class Program
{
    public static void Main()
    {
        Ucbucaq ob = new Ucbucaq();
        ob.teref1 = 5;
        ob.teref2 = 7;
        ob.teref3 = 8;

        double netice = ob.Sahe(); //(2)
        Console.Write("{0:##.#}", netice);

        Console.ReadKey();
    }
}
```

Ucbucaq sinfi içərisindəki Sahe() metoduna fikir verin. Bu metod müvafiq düsturla üçbucağın sahəsini hesablayır və nəticəni "sahe" adlı double tipindəki dəyişənə mənimsədir. (1) sətrində isə hesablanan bu qiymət geri qaytarılır. Buradakı return ifadəsinə diqqət edin. (2) sətrində isə metodu çağırdıq, bu nöqtədə metod icra olundu və geriyə double tipdə bir qiymət qaytardı. Qaytarılan bu qiyməti də çağrılma nöqtəsində "netice" adlı dəyişənə mənimsətdik. Beləliklə, proqramın nəticəsi aşağıdakı kimi olur:

```
Terefleri 5, 7, 8 olan ucbucagin sahesi: 17.3
```

Metodun qaytardığı qiymətin tipinin, metodun qaytarılma tipi ilə eyni olduğuna diqqət edin. Sahə dəyişəni double tipində olduğu üçün, metodun qaytarılma tipini də double seçdik. Əks halda, sintaktik xəta ilə qarşılaşacaqdıq.

## Parametrli metodlar

İndiyə qədər istifadə etdiyimiz metodlar, parametrsiz metodlar idi. Parametrli metodlar xaricdən bir və va bir neçə qiymət qəbul edə bilən metodlardır. Yəni bu cür metodları çağırarkən, onlara müəyyən qiymətlər ötürürük, metod bu qiymətlərə əsasən müəyyən bir əməliyyatı yerinə yetirir. Metoda göndərilən bu qiymətlərə arqument deyilir, bu arqumentləri özündə saxlayan dəyişənlərə isə metodun parametrləri deyilir. Parametrli metodların ümui sintaktik şəkli aşaöıdakı kimidir:

**<hüquq-təyinedici> <qaytarılma-tipi> <metod-adı>(tip parametr1, tip parametr2, ...)**

**{**

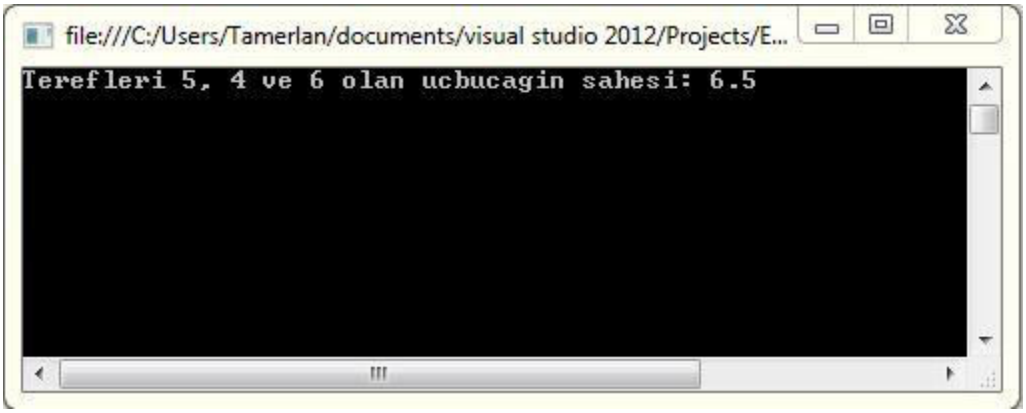
**kodlar...**

**}**

Parametrli metodların bir qiymət qaytarması zəruri deyil, bu tamamilə istəyə bağlıdır. Metodun adından sonra mötərizələr içərisində parametrlərə fikir verin, metoda istədiyimiz sayda parametr təyin edə bilərik, bu zaman parametrlər bir-birlərindən vergüllə ayrılmalıdır. Ucbucaq sinfindəki petodu parametrli şəklə optimizasiya edək. Belə ki, üçbucağın tərəflərini arqument kimi metoda ötürək və metod parametrlərinin qiymətləri ilə sahəni hesablasın:

```
using System;
class Ucbucaq
{
    public void Sahe(int t1, int t2, int t3)
    {
        int p = (t1 + t2 + t3) / 2;
        double sahe = Math.Sqrt(p * (p - t1) * (p - t2) * (p - t3));
        Console.Write("Terefleri {0}, {1} ve {2} olan ucbucagin sahesi: {3:##.#}", t1, t2,
t3, sahe);
    }
}
class Program
{
    public static void Main()
    {
        Ucbucaq ob = new Ucbucaq();
        ob.Sahe(5, 4, 6); //(1)
        Console.ReadKey();
    }
}
```

Bu proqramda Ucubucaq sinfi içərisindəki Sahe() metodu 3 ədəd t1, t2, t3 adlı parametrlərə malikdir. Metod, bu parametrlərin qiymətlərinə əsasən, sahəni hesablayacaq. Bu parametrlərə qiymətləri, metodu çağırma nöqtəsində ötürürük. (1) sətrinə fikir verin. Mötərizələrin içərisindəki 5, 4, 6 qiymətləri arqumentlərdir. Bu zaman, bu qiymətlər uyğun olaraq metodun parametrlərinə, yəni t1, t2, t3 dəyişənlərinə mənimsədilir. Beləliklə, proqramən nəticəsi aşağıdakı kimi olur:




Parametrli metodları çağırarkən, göndərilən arqumentlərin parametrlərin tipləri ilə üst-üstə düşdüyünü təmin etmək lazımdır. Əks halda bu sintaktik xəta olacaqdır. Məsələn, Sahe metodunu

ob.Sahe(5, 4.5, "Salam");

şəklində çağıra bilmərik. Çünki, sahə metodunun hər üç parametri tam tipdədir. Lakin, biz çağrılma nöqtəsində ikinci (4.5) və üçüncü ("Salam") arqumentlərinin tiplərini düzgün seçmədik. Parametrli metodların parametrləri, əslində metod daxilində lokal dəyişənlərdir. Onları, adi dəyişənlər kimi istifadə edirik. Göründüyü kimi bu metod, heç bir qiymət qaytarmır. Ona görə də, bu metodu məsələn,

double netice = ob.Sahe(5, 4, 6);

şəklində çağırsaq "Cannot implicitly convert type 'void' to 'double'" xətasını alardıq. Başqa bir nümunəyə baxaq. Bu nümunədə Sadedirmi() adlı bir metod təyin edəcəyik. Bu metod bir int tipdə parametrə malik olacaq və əgər metoda ötürülən arqument sadə ədəddirsə, metod geriyə true, əks halda false qiymətini qaytaracaq.

```
using System;
```

```
class Hesabla
{
    public bool Sadedirmi(int e)
    {
        bool netice = true;
        for (int i = 2; i < e / 2; i++)
        {
            if (e % i == 0)
            {
                netice = false;
                break;
            }
        }
        return netice;
    }
}
class Program
{
    public static void Main()
    {

        int eded = 17;
        Hesabla ob = new Hesabla();
        bool ok = ob.Sadedirmi(eded);
        if (ok == true) Console.WriteLine(eded + " ededi sadedir");
        else Console.WriteLine(eded + " ededi sade deyil");
        Console.ReadKey();
    }
}
```

Bu proqramda Hesabla adlı sinfin içərisində "Sadedirmi" adlı metod təyin olunub. Bu metod e parametrinin qiymətinin sadə olub olmamasını müəyyənləşdirir. Əgər ədəd sadədirsə, yəni dövr içərisindəki şərt heç vaxt doğru olmazsa, onda netice dəyişəni true olaraq qalır. Bir nöqtədə şərt doğru olarsa, onda netice dəyişəninə false qiyməti mənimsədilir və dövr dayandırılır. Bool tipdəki netice dəyişəninin qiyməti metod tərəfindən geri qaytarılır. Metodun qaytarılma tipinin də bool olduğuna fikir verin. Metodu çağıranda isə əgər ötürülən arqument sadə ədəddirsə, metod true qaytracaq. Beləliklə nəticə aşağıdakı kimi olacaq:

```
17 ededi sadedir
```

## return ifadəsi

Metodlar qitmət qaytararkən, **return** açar sözündən istifadə edildiyini görmüşdük. Amma return haqqında bilməli olduğumuz bəzi vacib şeylər var. Ümumiyyətlə, return bir proqram kontrol ifadəsidir. Bu kontrol ifadəsinin işi bir metodun işini dayandırmaqdan ibarətdir. Yəni bir metodun gövdəsində return varsa, proqram bu nöqtədən aşağıda qalan hissəni (metodun qapanış fiqurlu mötərizəsinə qədərki) icra etmədən ötürür. Ümumiyyətlə return ifadəsi ancaq qiymət qaytaran metodlarda istifadə olunmur. Qeyd olunduğu kimi, return metodların işini dayandırır. Əgər metod qiymət qaytarmırsa, onda return ifadəsi qarşısında heç bir şey yazılmır. Əks halda, return ifadəsinin qarşısında qaytarılacaq qiymət yazılır. Məsələn aşağıdakı proqramda "Metodum" adlı metod içərisindəki return ifadəsinə fikir verin:

```
using System;
class Sinif
{
   public void Metodum()
   {
      Console.WriteLine("Salam, necesen?");
      return;
      Console.WriteLine("Ne var ne yox?");
   }
}
class Program
{
   public static void Main()
   {

      Sinif ob = new Sinif();
      ob.Metodum();
      Console.ReadKey();
   }
}
```

Bu proqramda Metodum adlı metodun icrası zamanı proqramın axışı return ifadəsini gördükdən sonra, metodun işi bu nöqtədə dayandırılır və return – dan sonrakı heç bir kod icra olunmur. Belə kodlara *çatıla bilməyən kodlar* (Unreachable codes) deyilir. Beləliklə, bu proqramda ekrana sadəcə "Salam, necesen?" ifadəsi çıxır. Bu metodun heç bir qiymət qaytarmadığına və return ifadəsinin boş qeyd olunduğuna da fikir verin.

Qiymət qaytaran metodlarda isə return metodu yenə, metodun işini dayandırır, lakin əlavə olaraq metodun nəticəsi kimi geriyə bir qiymət ötürür. Çox vacib bir məqam var. Əgər bir metodun qaytarılma tipi **void** deyilsə, onda hökmən bu metod daxilində return ilə geriyə bir qiymət qaytarılmalıdır. Məsələn aşağıdakı proqramı kompilyasiya etmək

olmayacaq və bu bizə "'Sinif.Metodum()': not all code paths return a value" xətasını verəcək.

```
using System;
class Sinif
{
    public string Metodum()
    {
        Console.WriteLine("Salam, necesen?");
    }
}
class Program
{
    public static void Main()
    {

        Sinif ob = new Sinif();
        ob.Metodum();
        Console.ReadKey();
    }
}
```

Çünki, Metodum() geriyə string tipdə bir qiymət qaytaracaq şəkildə təyin olunub, amma heç return ifadəsi ilə heç bir qiymət qaytarılmır. Bu sintaksik xətadır. Digər tərəfdən, əgər bir metod bir qiymət qaytaracaq şəkildə (void olmayan şəkildə) təyin olunubsa, onda o metodun hökmən bir qiymət qaytarmasını təmin etmək lazımdır. Məsələn aşağıdakı proqrama baxaq:

```
using System;
class Sinif
{
    public string Insan(int eded)
    {
        if (eded > 0) return "Tamerlan";
    }
}
class Program
{
    public static void Main()
    {

        Sinif ob = new Sinif();
        ob.Metodum(5);
        Console.ReadKey();
    }
}
```

Bu proqramda Insan() metodu sadəcə parametrinin müsbət olduğu hallarda bir qiymət qaytarır. Yəni parametrə ötürülən arqument mənfi olarsa, metod geriyə bir qiymət qaytarmayacaq. Bu isə yenə bayaqkı sintaksik xətaya səbəb olacaq. Ona görə də, geriyə qiymət qaytaran metodun istənilən vəziyyətdə bir qiymət qaytarmasına zəmanət vermək lazımdır. Metodun gövdəsini aşaöıdakı kimi dəyişsək, xəta aradan qalxacaq:

```
public string Insan(int eded)
{
    if (eded > 0) return "Tamerlan";
    else return "Rustambayli";
}
```

Bu metod həmişə geriyə bir qiymət qaytarır. Əgər şərt düzgündürsə geriyə "Tamerlan", əks halda isə "Rustambayli" ifadəsi döndürülür. Başqa variant ola bilməz. Bununla belə, son olaraq qeyd etmək lazımdır ki, geriyə qiymət qaytaran metodların qaytardığı qiymətin tipi, geri dönüş tipi ilə eyni olmalıdır. Aşağıdakı metoda baxaq:

```
public string Insan(int eded)
{
    return true;
}
```

Metodun geri dönüş tipi string olduğu halda, bool tipdə qiymət qaytarmağa çalışdıq. Bu bir sintaktik xətadır.